September 1987

UILU-ENG-87-2258
CSG-71

COORDINATED SCIENCE LABORATORY
College of Engineering

MEASUREMENT-BASED RELIABILITY/ PERFORMABILITY MODELS

Mei-Chen Hsueh

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

MEASUREMENT-BASED RELIABILITY/PERFORMABILITY MODELS

BY

MEI-CHEN HSUEH

B.S., Providence College of Arts and Science, 1972
M.S., Portland State University, 1981

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1987

Urbana, Illinois

MEASUREMENT-BASED RELIABILITY/PERFORMABILITY MODELS

Mei-Chen Hsueh, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1987
Ravi Iyer, Advisor

This thesis describes measurement-based models based on real error-data collected on a multi-processor system. Models development from the raw error-data to the estimation of cumulative reward is described.

A workload/reliability model is developed based on low-level error and resource usage data collected on an IBM 3081 system during its normal operation in order to evaluate the resource-usage/error/recovery process in a large mainframe system. Thus, both normal and erroneous behavior of the system are modeled. The results provide an understanding of the different types of errors and recovery processes. The measured data show that the holding times in key operational and error states are not simple exponentials and that a semi-Markov process is necessary to model the system behavior. A sensitivity analysis is performed to investigate the significance of using a semi-Markov process, as opposed to a Markov process, to model the measured system.

A software reliability model is also developed based on low-level error data from the MVS operating system running on an IBM 3081 machine to describe the software error and recovery process. The semi-Markov model developed provides a quantification of system error characteristics and the interaction between different types of errors. As an example, we provide a detailed model and analysis of multiple errors, which constitute approximately 17% of all software errors and result in considerable recovery overhead.

PRECEDING PAGE BLANK NOT FILMED

In addition, a measurement-based performability model based on real error-data collected is proposed. A reward function, based on the service rate and the error rate in each state, is defined in order to estimate the performability of the system and, to depict the cost of different error types and recovery procedures.

ACKNOWLEDGEMENT

I am sincerely grateful to my thesis advisor, Ravi Iyer, for his guidance and encouragement. His assistance was of tremendous value. I would also like to give thanks to Professor Kishor Trivedi of Duke University for making available facilities at Duke and for valuable suggestions which provided additional breadth to this work.

Being a member of the Computer Systems Group has provided me with a continuous source of encouragement, suggestions, support and friendship. I would especially like to thank Pat Duba, Mark Sloan and Rene Llames for their valuable suggestions in the drafting of this thesis and Joe Rahmeh for his technical assistance.

Finally, I would like to thank my mother for the extensive care she provided my family, thereby making it possible for me to do my graduate work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1. Thesis Objectives

The development of realistic models to describe the error behavior of computer systems is a difficult problem. Although many researchers have addressed the modeling issue and have significantly advanced the state of the art, there is little or no validation of these models with field data. It is, therefore, extremely valuable to model the error and recovery process in a production system using real error data. Apart from providing useful information on how errors occur, this process also provides insight into the interaction between various system components. Additionally, it will be seen that it also allows explicit modeling of the relationship between resource usage and hardware and software errors, an area that has yet to be fully explored.

In this research we build a state-transition model which describes the resource-usage/error/recovery process of a computer system. This model is based on low-level error and resource usage data collected on a production system. The data were collected on an IBM 3081 system during its normal operation. Both the normal and erroneous behavior of the system are modeled. The results, therefore, provide an understanding of the different error and recovery processes and their relationship to various types of resource usage. Hardware and software reliabilities and their interaction are also modeled. Results show

that the error and recovery process on our measured system is best described by a semi-Markov process.

1.2. Related Research

The primary motivation for this research is that there has been no attempt to explicitly model the resource-usage/error/recovery process based on real data. The only research is that in [1,2], where the authors proposed the use of a double stochastic Poisson process to model a cyclic load-error relationship. The model assumes that the instantaneous error rate can be described by a cyclostationary Gaussian process (i.e., the workload has a cyclic pattern). Thus only the external behavior has been modeled. Furthermore, only a single workload variable (time spent in the kernel mode) was modeled.

Analytical models for hardware failure have been extensively investigated [3,4,5,6,7,8]. Although the time for different components to fail is usually assumed to be exponentially distributed, time-dependent failure rates and graceful degradation have been considered along with performability issues. Repairability has been modeled by Trivedi, et. al., [3,5,6,8], all of which assume constant repair times. A job/task flow based model is described in [9]. Failure occurrence is assumed to be a linear function of the service requests from a job/task flow. As shown in [10], the assumption of linearity may result in underestimating the effect of the workload, especially when the load is high.

Most software reliability models usually refer to the development, debugging and testing phases of the software as in [11,12] and [13,14]. Few of these models have been applied to the operational phase of the software. In [2] and [15], software failures in an operating environment are studied. Both studies found that at least 60% of system failures

are software related. Another study [16] shows that *undetected* software-related errors are due to either specification errors, implementation errors, or logic errors.

There is little explicit study of hardware/software reliability. The hardware/software interface is generally hard to model and experimental measurements are not easy to obtain and analyze. In [15], software failures in the operating system, which could be related to hardware problems, were analyzed and it was shown that errors in the hardware/software interface are often fatal. In [17], a methodology for joint hardware/software model construction and model processing using Stochastic Petri Nets is described.

With the exception of the software reliability growth models, which have been validated with real data, there are few, if any, models of software reliability in an operational environment. Exceptions include the hardware and software model discussed in [18] and a measurement-based model of workload dependent failures discussed in [10]. Both, however, only describe the external behavior of the system and do not provide insight into component level behavior.

It is therefore highly instructive to construct a detailed model based on low-level error data from a production system. Toward this end we have constructed a joint resource-usage/error/recovery model using error and resource usage data collected from an IBM system. The model provides detailed information on system behavior under normal and error conditions. Hardware and software failures of different severity are modeled. Multiple errors and the effect of on-line recovery routines are also considered.

1.3. Thesis Overview

A methodology for model construction based on real error data and resource usage information is described in Chapter 2. The model construction includes the resource usage (workload) characterization, error and recovery characterization, and modeling the overall system. For the workload characterization, we use a statistical clustering method to characterize the collected resource usages of the measured system from an n-tuple variable of infinite points into a few number of sets. Thus, a state-transition model of resource usages of the system is constructed based on these sets.

Different types of component errors and recovery procedures are also described in detail and classified in Chapter 2. A two-level error data reduction scheme is employed to identify individual error incidents and ensure that the analysis is not biased by error records relating to the same problem. The interaction of hardware and software errors is modeled in this chapter. The three models describing resource usage, error and recovery are then combined to form an overall model. The conditional transition probabilities as well as the sojourn times of states are estimated from real data. Results show that the resource-usage/error/recovery process is a semi-Markov process.

In Chapter 3 we perform four different kinds of model analyses to show the characteristics of the measured system. First, we use the model built in Chapter 2 to evaluate key characteristics of the system, such as the state occupancy probability and the unconditional transition probability from one specified state to another. These measures provide us with a very fair estimation of the model behavior. Second, we estimate the error probability due to the workload from the model. The analysis shows that the error probabilities appear to be not only a function of the resource usage, but are also related to the length of the sojourn time in a resource usage state. Third, the model validation is performed by

comparing the results predicted from the model with the values estimated from the actual observations. Finally, we perform an analysis to investigate the significance of using a semi-Markov process, as opposed to a Markov process, to model the measured system.

In Chapter 4 a measurement-based software reliability model is built. In addition to describing the software error and recovery process in the measured system this model also provides a quantification of software system error characteristics and the interaction between different types of software errors.

A performability model based on real data is proposed in Chapter 5. A reward function, based on the service rate and the error rate in each state, is defined in order to estimate the performability of the measured system and to depict the cost of different error types and recovery procedures. The conversion of a semi-Markov model to its Markov version is also demonstrated in this chapter. This conversion gives us the ability to use an existing system performance estimator to estimate the performability of the measured system.

In Chapter 6 we provide a summary of this research and highlight some important conclusions drawn from this work.

CHAPTER 2

RESOUCE-USAGE/ERROR/RECOVERY MODELING

2.1. Workload Modeling

In this section we build a state-transition model to describe the variation in system activity. It will later be shown that this approach allows an error to be considered as a transition from normal activity. System activity is characterized by a number of resource usage parameters. A statistical clustering technique is employed to reduce the potential many to many transitions of the workload vector to a small number of states representative of those found in the data. The data for our studies came from an IBM 3081 system running the MVS operating system. The system consists of dual processors with two time-multiplexed channel sets. Together these two sets allow a maximum of 24 subchannels to be simultaneously active in each I/O cycle.

2.1.1. Resource Usage Characterization

The workload data was collected using the IBM MVS/370 system Resource Management Facility (RMF) [19]. RMF is a flexible tool for measuring the performance of an IBM system. It measures data in two ways: by exact count and by sampling. The exact count method checks the appropriate system indicators at the beginning and the end of an interval and calculates the difference. The sampling method checks the appropriate system indicators at each cycle within an interval (e.g., an interval may be one hour and a cycle may be

500 milliseconds). At the end of the interval the mass of data collected at each cycle is reduced to either minimum, maximum, and average values or to a percentage value. The results presented here are based on three months of sampled RMF data, with a cycle time of 500 milliseconds and an interval of one hour.

Four different resource usage measures were selected to represent the workload of three basic components of the computer hardware system.

CPU - fraction of the measured interval for which the CPU is executing instructions

CHB - fraction of the measured interval for which the channel was busy and the CPU was in the wait state (this parameter is usually used to measure the degree of contention in our system)

SIO - number of successful Start I/O and Resume I/O instructions issued to the channel

DASD - number of requests serviced on the direct access storage devices

Although several other measures were available, we decided to use only the measures listed above so as to keep the model trackable. The methodology presented here is easily extended to incorporate other measures.

2.1.2. Workload Clustering

At any interval of time the measured workload is represented by a point in 4-dimensional space, (CPU, CHB, SIO, DASD). Cluster analysis is used to divide the workload into similar classes according to a pre-defined criterion.[1] This allows us to concisely describe the dynamics of system behavior and extract a structure that already exists in the

[1]Potentially, we can have an uncountably large number of points in the workload space. Intuitively, only a countable number of combinations of four measures do in fact occur. Further, it is seen that they usually occur in clusters.

workload data.[2] Each cluster (defined by its centroid) is then used to depict a system state and a state-transition diagram (consisting of inter-cluster transition probabilities and cluster sojourn times) is developed.

A k-means clustering algorithm [21,22] was used for cluster analysis. Briefly, the algorithm partitions an N-dimensional population into k sets on the basis of a sample. It starts with k groups each of which consists of a single random point. Each new point is added to the group with the closest centroid. After a point is added to a group, the mean of that group is adjusted in order to take the new point into account. This process is repeated until the changes in the cluster means become negligibly small. Thus at each stage the k-means are, in fact, the means of the groups they represent. Therefore, k non-empty clusters, $C_1, C_2, \ldots, C_k$, are sought such that the sum of the squares of the Euclidean distances of the cluster members from their centroids is minimized, i.e.,

$$\sum_{j=1}^{k} \sum_{i} \| x_i - \bar{x}_j \|^2 \quad \rightarrow \quad \text{minimum}$$

where $x_i \in C_i$ and $\bar{x}_j$ is the centroid of cluster C_j.

Two types of workload clusters were formed. In the first case CPU and CHB were selected to be the workload variables. This combination was found to best describe the CPU-bound load (nearly 60% of the observations have a CPU usage greater than 0.72). In the second case the clusters were formed considering SIO and DASD as workload variables. This combination was found to best describe the I/O workload. Table 2.1 shows the results for these two cases.

An examination of Table 2.1 also shows the dynamics of the measured system behavior. We see in Table 2.1(a) that about 36% of the time the CPU is highly loaded

[2] Similar clustering techniques are also used for workload characterization in [20].

Table 2.1. Characteristics of workload clusters

(a) CPU workload

Cluster id	% of obs	Mean of CPU	Mean of CHB	Std dev of CPU	Std dev of CHB
W_1	7.44	0.0981	0.1072	0.0462	0.0436
W_2	0.50	0.1126	0.5525	0.0433	0.0669
W_3	2.73	0.1547	0.2801	0.0647	0.0755
W_4	12.41	0.3105	0.1637	0.0550	0.0459
W_5	0.74	0.3639	0.3819	0.0365	0.1923
W_6	17.12	0.5416	0.1287	0.0560	0.0511
W_7	22.58	0.7207	0.0848	0.0576	0.0301
W_8	36.48	0.9612	0.0168	0.0362	0.0143
R^2 of CPU =	0.9724				
R^2 of CHB =	0.8095				
overall R^2 =	0.9604				

(b) I/O workload

Cluster id	% of obs	Mean of SIO	Mean of DASD	Std dev of SIO	Std dev of DASD
U_1	8.89	16.80	0.95	6.80	1.30
U_2	36.05	41.59	2.99	7.51	1.92
U_3	1.48	44.37	20.62	8.55	4.18
U_4	1.73	60.07	38.84	6.77	8.42
U_5	42.72	67.34	5.19	7.92	3.72
U_6	0.49	87.30	31.19	3.87	9.84
U_7	7.9	96.20	6.02	8.73	3.34
U_8	0.74	141.10	10.10	10.28	8.50
R^2 of SIO =	0.8861				
R^2 of DASD =	0.7176				
overall R^2 =	0.8751				

(0.96) and almost 76% of the time the CPU load is above 0.5. Since the measured system is a two-processor machine, we may say that 76% of the time at least one of the processors is busy. Note that, with increasing CPU usage, CHB (CPU wait and channel busy) decreases. This indicates that resource contention is not a problem in our measured system. In Table 2.1(b) (the I/O load), both clusters U_2 and U_3 have a very close channel start I/O rate (SIO) but the disk service rate (DASD) of U_3 is as much as 10 times that of U_2. This indicates that some I/O requests result in a burst of data while the others only in a few words. A burst transfer however occurred only 4% of the time ($U_3 + U_4 + U_6$). This result may be due to the fact that our measurements were made during work hours, but I/O-bound jobs are normally executed during off-work hours.

2.1.3. Resource Usage Model

State-transition diagrams of these two different types of workload clusters are shown in Figure 2.1 and Figure 2.2. The transition probabilities from state i to state j, p_{ij}, are estimated from the measured data using:

$$p_{ij} = \frac{\text{observed number of transitions from state } i \text{ to state } j}{\text{observed number of transitions from state } i}. \qquad (2.1.1)$$

These two figure provide us with not only the details of workload dynamics but also the interactions among clusters. Figure 2.1 shows that once the CPU load reached 0.5 (W_6), the transition of the greatest probability was to its next higher load (W_7) and the transition to its next lower load ($W_{4,5}$) occurred with the second greatest probability. This can be seen in states W_6, W_7, and W_8. However, when the CPU load is low (i.e., less than 0.5), the change to a higher load is much faster. For example, with 0.333 probability the CPU load changed from W_1 to $W_{4,5}$ and 0.424 probability from $W_{4,5}$ to W_7.

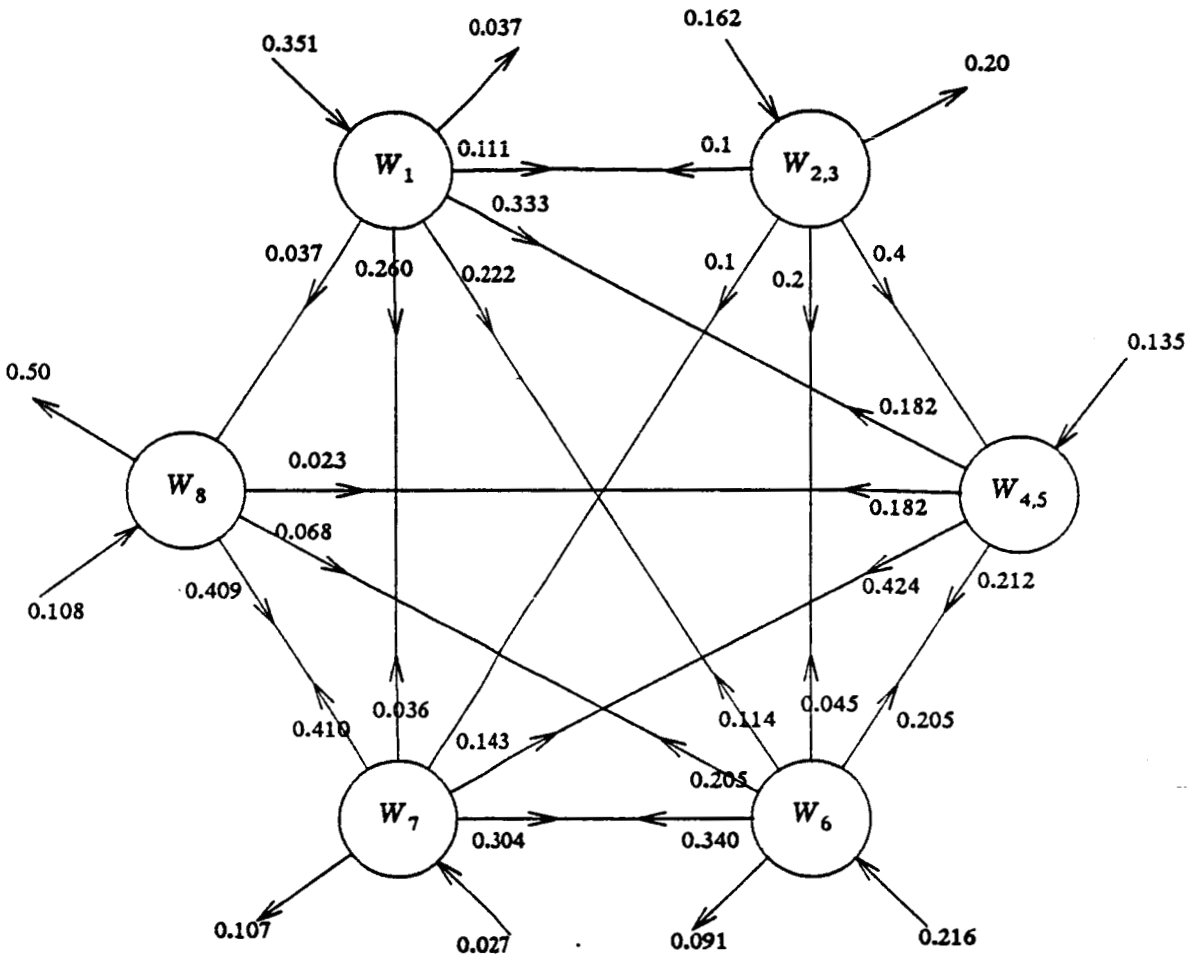


Figure 2.1. State-transition diagrams of CPU bound load

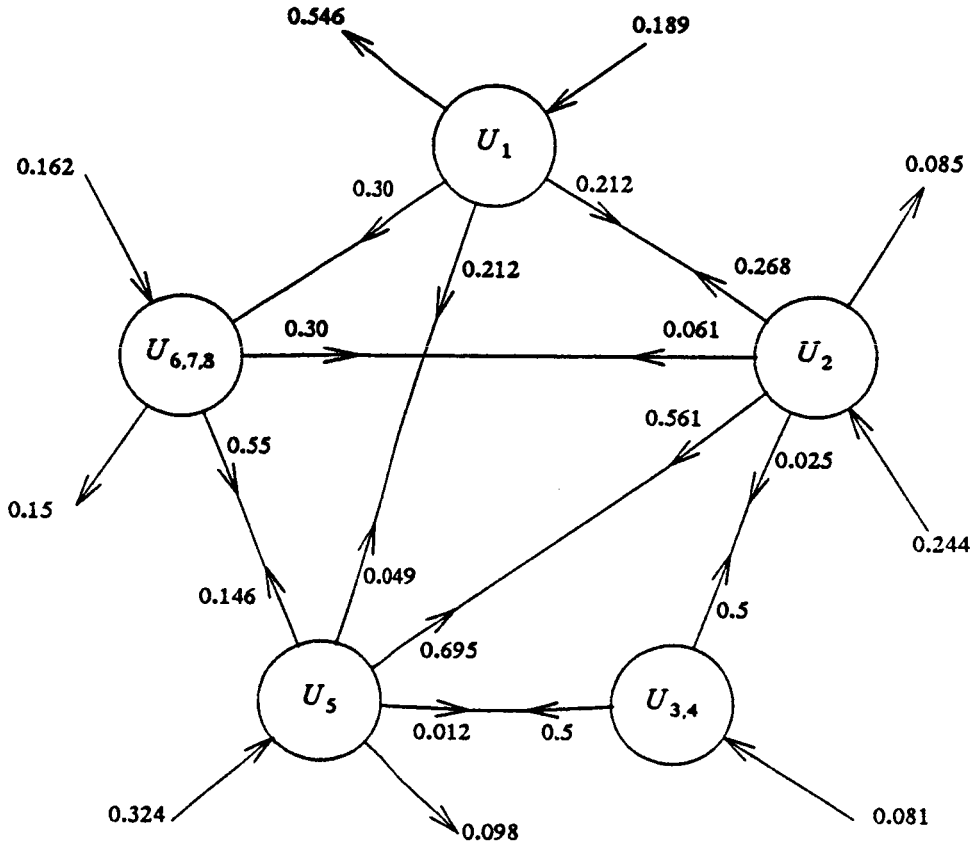


Figure 2.2 State-transition diagrams of I/O bound load

Figure 2.2 shows the transitions among various I/O loads. This figure confirms our previous observation that most often the I/O workload fluctuates back and forth between two moderate levels. U_2 and U_5 (0.69 and 0.56) and that there are occasional requests for burst I/O (0.025 from U_2 and 0.012 from U_5 to $U_{3,4}$).

2.2. Error Modeling

In this section the collection and characterization of errors is discussed. A state-transition diagram to describe different error states is developed. The measured system incorporates built-in error detection facilities, and many components also provide for recovery through retry or redundancy. The error and recovery information is logged into a permanent data set called LOGREC [23]. For each error, whether recoverable or not, the operating system creates a time-stamped record describing the error and providing relevant information on the state of the machine. In each record there are a number of bits describing the type of error, its severity, and the result of hardware and software attempts to recover from the problem. From this data six different types of errors were collected :

(1) CPU-related errors	- those that affect the normal operation of the CPU; the errors may originate in the CPU itself, in the main memory, or in a channel.
(2) Temporary channel errors	-those that are recovered by channel retry and do not result in the termination of the channel control program.
(3) Temporary (soft) disk errors	- those I/O errors that are recovered by correcting the data or by retrying the hardware instruction.
(4) Temporary (hard) disk errors	- those I/O errors that are recovered by software instruction retry or by a functional recovery routine(s).
(5) Permanent disk errors	- those I/O errors that are not correctable and can not be recovered by retrying the operation, and
(6) Software errors	- software incidents that are due to invalid supervisor calls, program checks and other software exception conditions.

2.2.1. Error Clustering

Due to the manner in which errors are detected and reported in a computer system, it is possible that a single fault may manifest itself as more than one error, depending on the activity at the time of the error. The different manifestations may not all be identical [24]. The system recovery usually treats these errors as isolated incidents. In order to address this problem and to ensure that the analysis is not biased by error records relating to the same problem, two levels of data reduction were performed.

First, a coalescing algorithm described in [10] was used to analyze the data and merge observations which occur in rapid succession and relate to the same problem. Next, a technique described in [24] to automatically group records most likely to have a common cause, was used (See Appendix A for the details).[3] By using these two methods, we classified errors into five different classes. These classes are called error events since they may contain more than one error and are defined as follows.

CPU : that caused errors to be logged as CPU-related errors

CHAN : that caused errors to be logged as channel errors

SWE : that caused errors to be logged as software errors

DASD : that caused errors to be logged as direct access storage device errors

MULT : that caused errors affecting more than one type of component

Table 2.2 lists the frequencies of different types of errors. In this table we found that about 80% of errors are disk and software errors. We also note that about 17% of the errors are classified as multiple errors (MULT). A MULT error is mostly due to a single cause but the fault has non-identical manifestations provoked by different types of system activity.

[3]Although this second reduction is not essential to this work, it allows us to notice several multiple errors which otherwise would not have been noticed.

Since the manifestations are non-identical, recovery may be complex and hence imposes considerable overhead on the system. It should be noted that such an error event (17% of our data) has not been modeled before.

2.3. Recovery Modeling

When an error is detected in the measured system, an appropriate recovery routine is invoked depending on the severity of the error. The recovery procedures were divided into four categories in increasing order of recovery cost. The recovery cost was measured in terms of the system overhead required to handle an error. The lowest level (hardware recovery), involves the use of an error correction code (ECC) or hardware instruction retry and has minimal overhead. If hardware recovery is not possible (or unsuccessful), the next level, i.e., software controlled recovery, is invoked. This could be simple, e.g., terminating the current program or task in control, or complex, e.g., invoking a specially designed recovery routine(s) to handle the problem. The third level of recovery (ALT) involves transferring the tasks to a functioning processor(s) when one of the processors experiences

Table 2.2. Frequency of errors

Type of error	Frequency	Percent
CPU	2	0.04
CHAN	119	2.23
MULT	924	17.33
SWE	1923	36.07
DASD	2364	44.34
total	5332	100.00

an un-recoverable error. If no on-line recovery is possible, the system is brought down for off-line repair. Figure 2.3 shows a flow chart of the recovery process. Table 2.3 lists the distribution of recovery levels. From Table 2.3 we note that about 73% of errors were

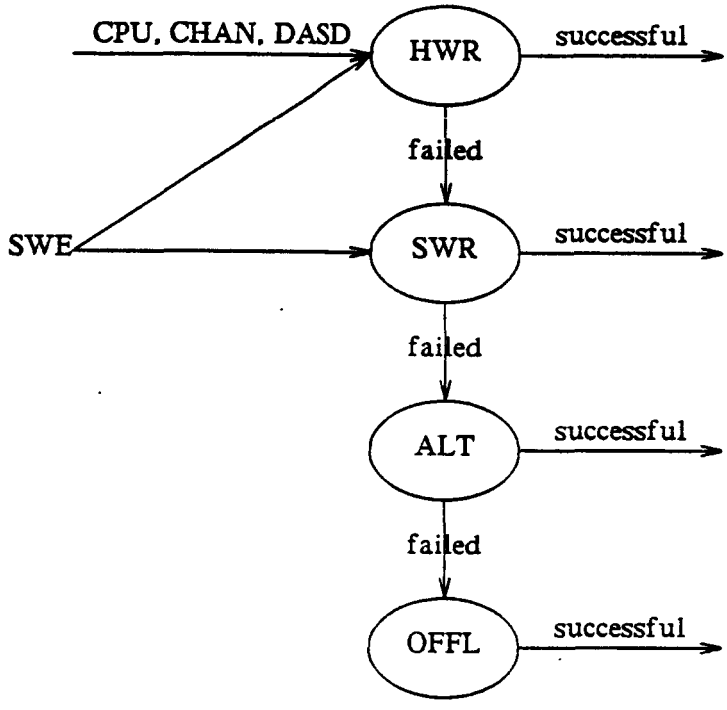


Figure 2.3. Flow chart of recovery processes

Table 2.3. Percentage distribution of recovery procedures

Recovery Procedure	Percent
HWR	73.35
SWR	26.56
ALT	0.02
OFFL	0.07

successfully handled through hardware recovery and most of the others were recovered from by use of the software recovery procedure.

2.4. Resource-Usage/Error/Recovery Model

In this section we combine the separate workload, error and recovery models, developed so far, into a single model shown in Figure 2.4. A null state W_0 is added to

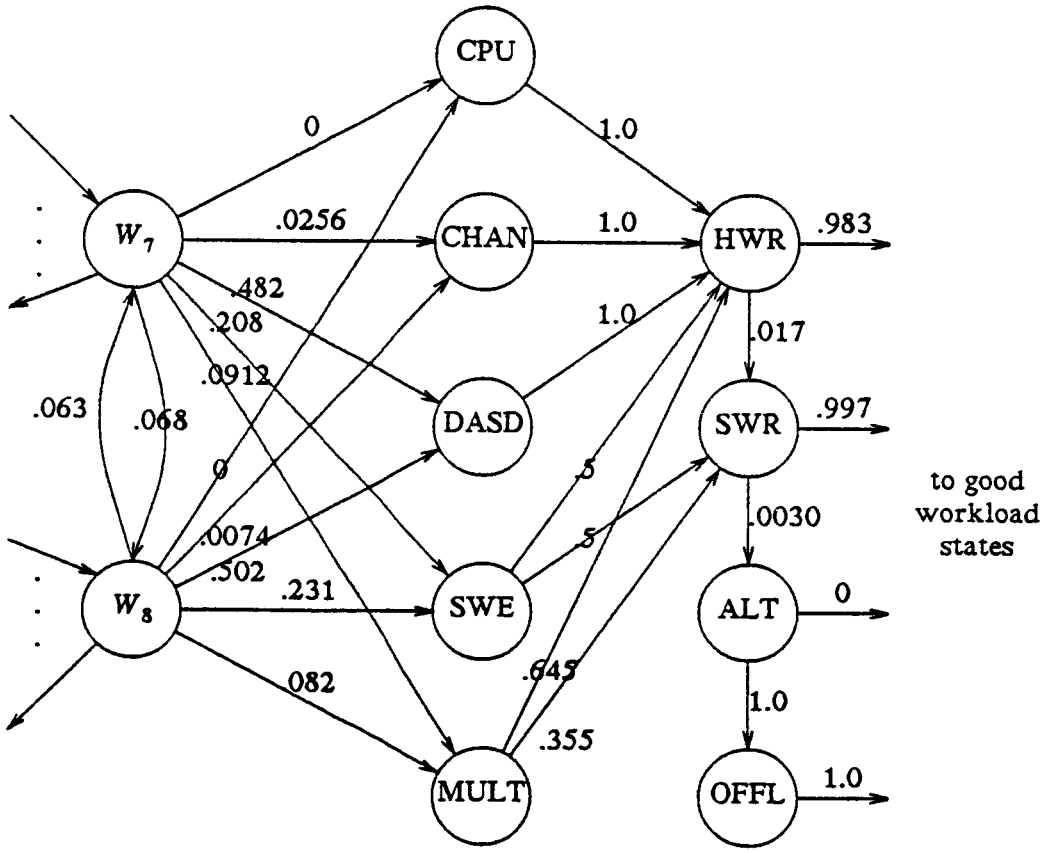


Figure 2.4. State-transition diagram of resource-usage/error/recovery model

represent the state during the non-measured period although it is not shown in Figure 2.4. The transition probabilities among states are estimated from the measured data using Equation 2.1.1. Notice that, unlike other models this describes both the normal and erroneous behavior of the system. The model has three different classes of states: normal operation states (S_N), error states (S_E), and recovery states (S_R). Note that the normal state has two different types of transitions: the first, to other normal states and the second, to error states.

Under normal conditions, the system makes transitions from one workload state to another. The occurrence of an error results in a transition to one of the error states. The system then goes into one or more recovery modes after which, with a high probability, it returns to one of the "good" workload states. The state-transition diagram of Figure 2.4 shows that nearly 98.3% of the hardware recovery requests and 99.7% of the software recovery requests are successful. Thus the error detection, fault isolation and on-line recovery mechanism allow the measured system to handle an error efficiently and effectively. In only less than 1% of the cases is the system not able to recover.

Figure 2.5 shows the state-transition diagram of a MULT error (a MULT event), i.e., given that a multiple error has occurred. The model shows that disk and software errors are strongly correlated in multiple errors. From the diagram, it is seen that in about 65% of the cases a multiple error starts as a software error (SWE) and in 32% of the cases it starts as a disk error (DASD). Given that a disk error has occurred there is nearly a 30% chance that a software error will follow. It is also interesting to note that there is a 64% chance that one software error will be followed by another different software error.

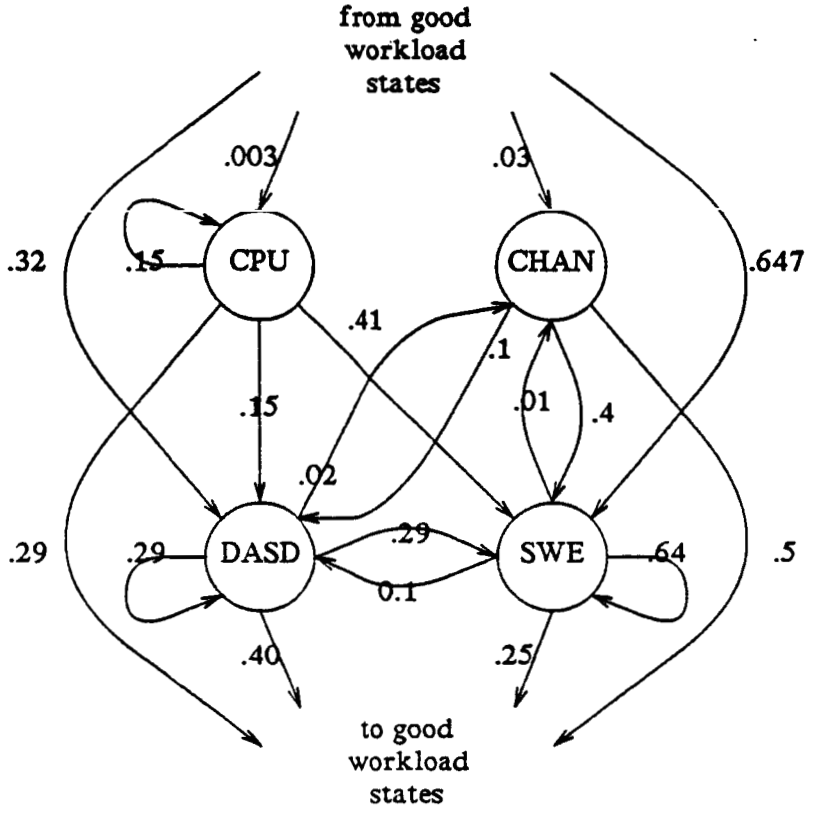


Figure 2.5. State-transition diagram for multiple errors (MULT)

2.4.1. Waiting and Holding Time Distributions

We used the state-transition diagram to show the relationship among the workload, error, and recovery processes in the measured system. We also showed the interactions among the errors. In this subsection we will present the characteristics of the measured system in terms of the state waiting and holding times.

The waiting time for state i is the time that the process spends in state i before making a transition. The holding time for a transition from state i to state j is the time that the process spends in state i before making a transition to state j [25]. Table 2.4 shows the mean waiting times of both the workload and error states. It is well-known that the mean and standard deviation of an exponential distribution are the same. Thus an examination of the mean and standard deviation of the waiting times in Table 2.4 appears to indicate that not all waiting times are simple exponentials. This is particularly pronounced in Table 2.4(c) which refers to the error states.

Figure 2.6 shows the densities of waiting and holding times for one of the CPU load states, W_8 (see Appendix B for all states). Figure 2.6(a) shows the waiting time for W_8, and Figure 2.6(b) and 2.6(c) represent the holding times from state W_8 to DASD and SWE error states. These densities are fitted to phase-type exponential density functions [26].

$$f(t) = \sum_{i=1}^{n} a_i g_i(t) ,$$

where $a_i \geq 0$, $\sum_{i=1}^{n} a_i = 1$, and n is the number of phases. The $g_i(t)$ function can be a simple exponential, a multi-stage hyperexponential, or a multi-state hypoexponential density function. The definitions of these three types of exponential functions are listed below.

(1) Exponential: $g(t) = \lambda e^{-\lambda t}$.

(2) Hyperexponential: $g(t) = \sum_{i=1}^{r} \alpha_i \lambda_i e^{-\lambda_i t}$, where $\lambda_i > 0$, $\alpha_i \geq 0$, and $\sum_{i=1}^{r} \alpha_i = 1$.

(3) Hypoexponential: $g(t) = \sum_{i=1}^{r} a_i \lambda_i e^{-\lambda_i t}$, where $\lambda_i > 0$, $\lambda_i \neq \lambda_j$ if $i \neq j$, and

$$a_i = \prod_{\substack{i=1 \\ i \neq j}}^{r} \frac{\lambda_j}{\lambda_j - \lambda_i}.$$

Table 2.4 Mean waiting time (in seconds) of states

(a). CPU bound workload states

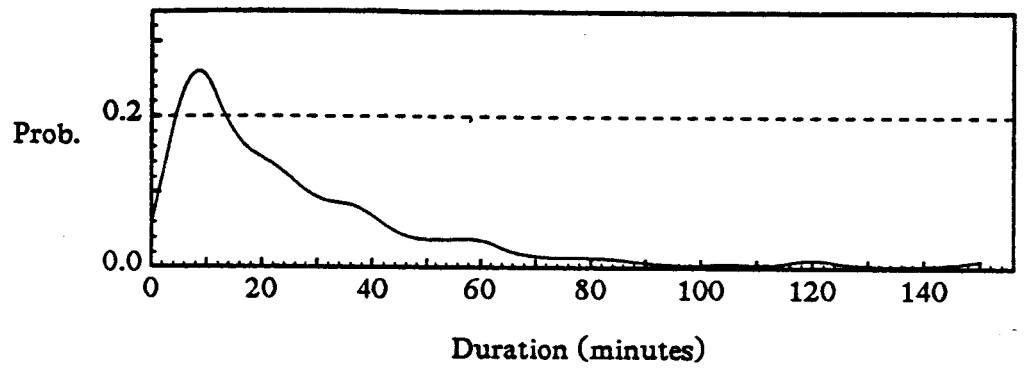
State	# of obs	Mean waiting time	Standard deviation	Std error of mean
W_1	53	1263.71	1384.20	190.13
W_2	2	289.65	1.19	0.84
W_3	20	698.79	913.30	204.22
W_4	130	1203.05	1130.28	99.13
W_5	11	613.74	421.73	127.16
W_6	147	1380.86	1588.76	131.04
W_7	268	1071.31	1004.46	61.36
W_8	266	1612.72	2576.35	157.97

(b). I/O workload states

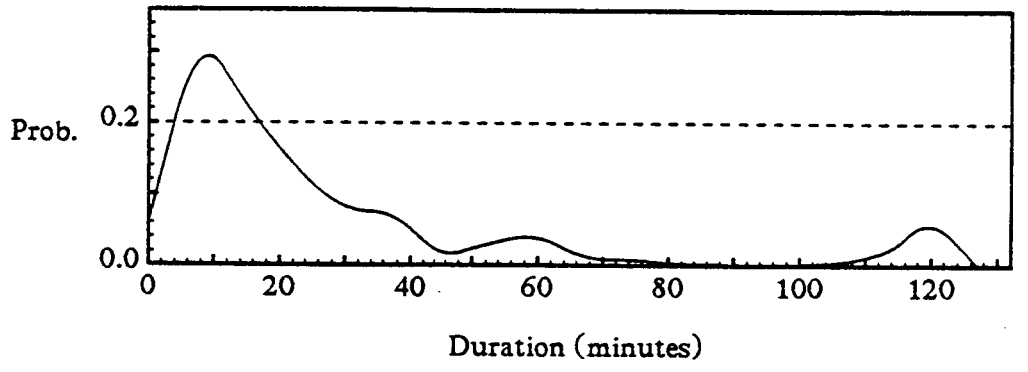
State	# of obs	Mean waiting time	Standard deviation	Std error of mean
U_1	45	1221.75	1475.70	219.98
U_2	316	1453.19	1530.75	86.11
U_3	12	1437.15	1452.86	419.40
U_4	18	1137.41	616.67	145.39
U_5	420	1243.63	1550.49	75.66
U_6	4	1696.85	1540.19	770.10
U_7	86	937.45	1127.57	121.59
U_8	9	387.74	176.96	58.99

(c). Error states

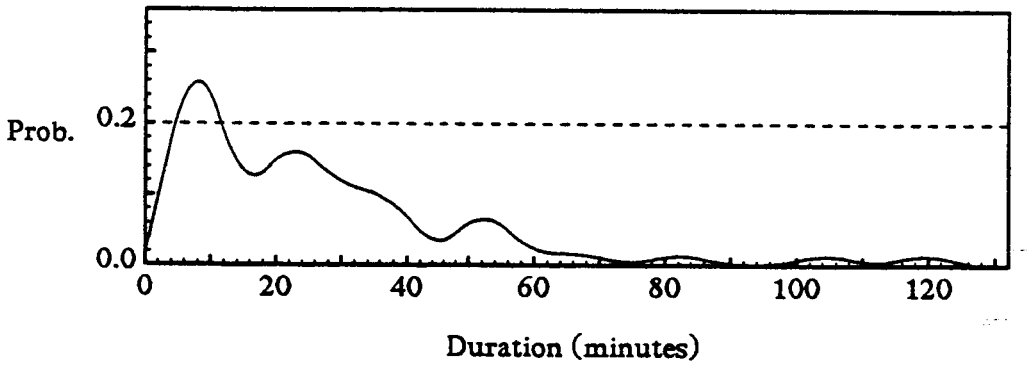
State	# of obs	Mean waiting time	Standard deviation	Std error of mean
CHAN	13	5.08	18.31	5.08
SWE	201	41.35	103.35	7.29
DASD	401	120.86	223.89	11.18
MULT	77	293.28	262.84	29.95



(a) Waiting time density of state W_8 (CPU = 0.96)



(b) Holding time density from state W_8 to state DASD



(c) Holding time density from state W_8 to state SWE

Figure 2.6. Waiting and holding time densities

Thus the graphs in figure 2.5 were fitted to the following functions (tested by using the Kolmogorov-Smirnov test [26] at the 0.01 significance level).

(1). waiting time : $f(t) = 0.000146e^{-0.002t} + 0.000939e^{-0.00103t} + 0.000033e^{-0.0002102t}$

(2). to a DASD error : $f(t) = 0.00094e^{-0.004t} + 0.0008355(e^{-0.000937t} - e^{-0.006595t})$

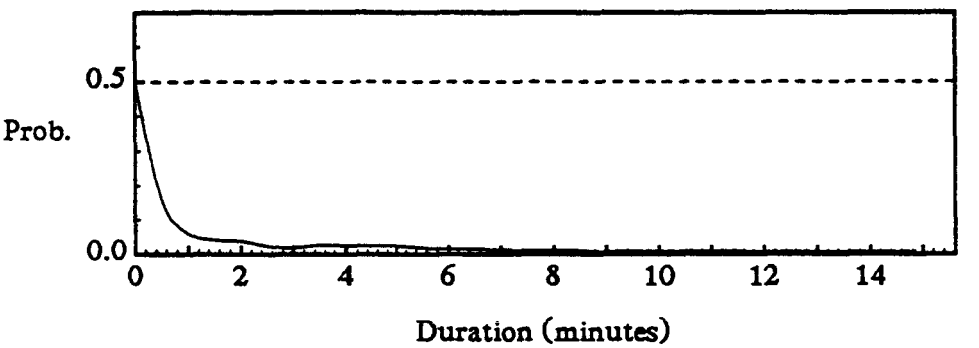
(3). to a SWE error : $f(t) = 0.00085e^{-0.005t} + 0.000701(e^{-0.000716t} - e^{-0.004688t})$

2.4.2. Recovery Distributions

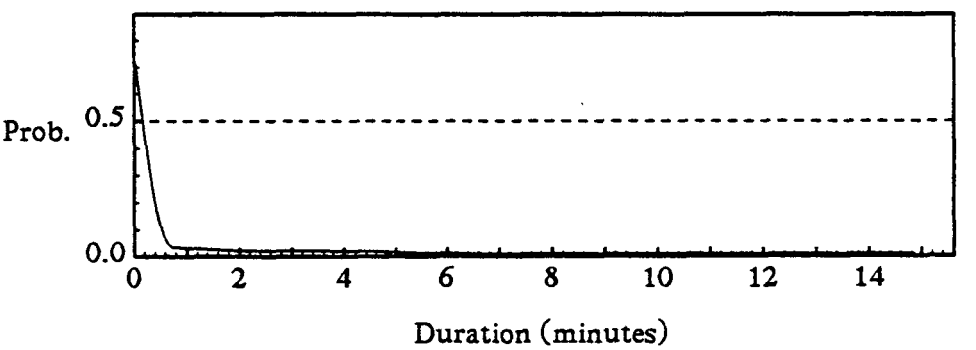
In our data, the selection of the destinations from any state of S_R was found to be independent of the holding time distribution. Further, for our system the time taken for each type of recovery can reasonably be considered constant. The overall recovery time, i.e., the duration of an error event (or the holding time in an error state), however was not constant since an error event may involve more than one recovery attempt. This time is computed as the time difference between the first detected error and the last detected error caused by the same event. The duration of an error event can be used to measure the effectiveness of recovery from this event and also the severity of error. Figure 2.7 shows examples of error duration densities for three different types of errors. Again, the following phase-type exponential densities were fitted to the graphs shown in Figure 2.7 (tested at the 0.01 significance level).

(1) DASD : $f(t) = 0.0375e^{-0.15t} + 0.007e^{-0.1t} + 0.008635e^{-0.014562t} + 0.0001861e^{-0.0021377t}$

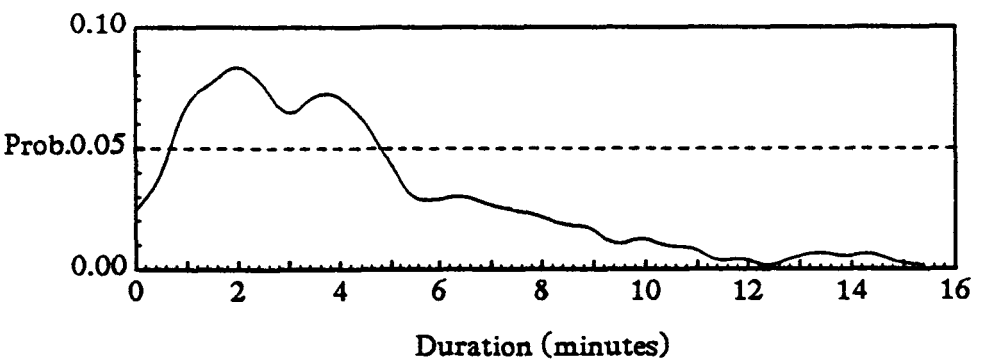
(2) SWE : $f(t) = 0.041181e^{-0.044518t} + 0.0002704e^{-0.0036075t}$



(a) DASD error duration density



(b) SWE error duration density



(c) MULT error duration density

Figure 2.7. Error duration densities

(3) MULT: $f(t) = 0.004371(e^{-0.003817t} - e^{-0.0301092t})$

2.5. Summary

In summary, we have developed a state-transition model which describes the normal and error behavior of the system. Some key characteristics of the model are:

(1) workload dynamics are explicitly described,

(2) error/recovery is explicitly described,

(3) waiting times in some workload and in most error states can not be modeled as simple exponentials, and

(4) the holding times from a given workload state to various error states are dependent on the destinations.

Thus, the resource-usage/error/recovery process is modeled as a complex irreducible semi-Markov process with the state OFFL as recurrent, making the overall model ergodic. Furthermore, the process is not an independent semi-Markov process since the waiting and holding time distributions are distinct for some states.

CHAPTER 3

MODEL ANALYSIS

Now that we have an overall model, we show the usage of this model to predict key system characteristics. The mean time between different types of errors is evaluated along with model characteristics such as the occupancy probabilities of key error and workload states. Since the normal state transitions are also available, we can explicitly examine those states which are crucial from a error viewpoint. In order to evaluate the model behavior, the model parameters, however, have to be defined and then the derivations of the measures can be carried out. Thus in the next section we provide the definitions of the model parameters and the derivations of some important measures.

3.1. Model Parameters

From Chapter 2 we know that the measured system is best modeled as a semi-Markov process. Assume that M is an n-state semi-Markov model and given a stochastic transition probability matrix $P = \left[p_{i,j}\right]$, $p_{i,j} \geq 0$, $i=1,2,....n$, $j=1,2,....n$, $\sum_{j=1}^{n} p_{i,j}=1$, and a holding time density function matrix $H(t) = \left[h_{i,j}(t)\right]$, $t \in (0,\infty)$, the mean holding time of the process staying in state i before making transition to state j, $\bar{\tau}_{i,j}$, is

$$\bar{\tau}_{i,j} = \int_{t=0}^{\infty} t\, h_{i,j}(t)dt \ . \tag{3.1.1}$$

We mentioned that in Section 2.4.1 the waiting time for state i is the time that the process spends in state i before making a transition. Thus, a waiting time is merely a holding time that is unconditional on the destination state. Hence the mean waiting time $\bar{\tau}_i$ is related to the mean holding time $\bar{\tau}_{i,j}$ by

$$\bar{\tau}_i = \sum_{j=1}^{n} p_{i,j}\bar{\tau}_{i,j} \ . \tag{3.1.2}$$

Suppose that a process has been operating unobserved a long time and given that the process is now making a transition, the probability that the transition is to state j, π_j, must satisfy n simultaneous equations

$$\pi_j = \sum_{i=1}^{n} \pi_i p_{i,j} \ . \tag{3.1.3}$$

We note that these n equations are linearly dependent. This linear dependency can be easily shown by summing these n equations, which results in $1 = 1$. Therefore no unique solution for π_j can be obtained from just by solving the equations (3.1.3). Since we know that the probabilities that the transition to all states have to sum to one, i.e.,

$$\sum_{i=1}^{n} \pi_i = 1 \ . \tag{3.1.4}$$

Then we can use Equation 3.1.3 in conjunction with Equation 3.1.4 to provide an unique solution for the steady state transition probability. After we substitute Equation 3.1.4 into the left hand side of Equation 3.1.3, we have

$$1 = \pi_j p_{j,j} + \sum_{\substack{i=1 \\ i \neq j}}^{n} \pi_i (1 + p_{i,j}). \tag{3.1.5}$$

The unique solution for π_i can be obtained by solving n linear equations of (3.1.5). The matrix form of the solution is

$$\pi = O \left[U - I + P \right]^{-1}. \tag{3.1.6}$$

where π, O, U and I are:

(1) $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$,

(2) O is an unit row vector, i.e., all elements are one,

(3) U is an unit matrix, and

(4) I is an identity matrix.

After deriving the steady state probability (also called limiting state probability), the probability of a state being occupied by the process and the probability of the process entering a specified state can be obtained accordingly.

The steady state occupancy probability of state j, denoted as Φ_j, is the probability that the process occupies state j when the system reaches a stable stage, and is evaluated as [25]:

$$\Phi_j = \Phi_{i,j} = \frac{\pi_j \bar{\tau}_j}{\bar{\tau}}. \tag{3.1.7}$$

where $\bar{\tau} = \sum_{i=1}^{n} \pi_i \bar{\tau}_i$.

We are sometimes interested not only in the probability that the process will occupy a state at some time in the future but also in the probability that the process will enter a

state at some particular future time. Thus, the probability that the process is just entering state j at some time instant after the system is in the steady state, e_j, is just the state occupancy probability Φ_j divided by its mean waiting time $\bar{\tau}_j$ [25]:

$$e_j = e_{i,j} = \frac{\Phi_j}{\bar{\tau}_j} . \tag{3.1.8}$$

After substituting the result of Equation 3.1.7 into Equation 3.1.8 we have

$$e_j = \frac{\pi_j}{\bar{\tau}} . \tag{3.1.9}$$

In a semi-Markov process — as in every life — an important question is "How long does it take to get from here to there?". Assume that the time it takes to reach state j for the first time if the system is in state i at time zero is $\Theta_{i,j}$, then $f_{i,j}(t)$, the probability that $\Theta_{i,j} = t$, is defined as [25]:

$$\begin{aligned} f_{i,j}(t) &= \text{Pro}\,(\Theta_{i,j} = t) \\ &= \sum_{\substack{r=1 \\ r \neq j}}^{n} p_{i,r} \int_0^t h_{i,r}(\sigma) f_{r,j}(t-\sigma) d\sigma + p_{i,j} h_{i,j}(t) . \end{aligned} \tag{3.1.10}$$

The process can make transitions to other states before it first reaches state j at time t, or it may stay in state i and then make a direct transition to state j at time t. The first term of the right hand side of Equation 3.1.10 computes the probability of being in state i for any $\sigma \epsilon [0, t)$ and the probability of the process being in another state r at the beginning time of $t-\sigma$ after the process is out of state i. The second term computes the probability if the process makes a transition directly from state i to state j at time t. Therefore, the time to move from state i to state j can be estimated as the mean first passage time for a process from state i to j, and it is evaluated as

$$\bar{\Theta}_{ij} = \int_0^\infty t f_{ij}(t)dt \ . \tag{3.1.11}$$

Since the $f_{ij}(t)$ is a recursive function which is shown in Equation 3.1.10, the computation time increases exponentially as t increases. However, in statistical the mean of a random variable can be estimated as the first moment of its moment generating function, e.g., the first derivative of its exponential transformation at point zero. The exponential transformation of a function $g(t)$, denoted as $g^e(s)$ is defined as

$$g^e(s) = \int_0^\infty g(t)e^{-st}dt \ . \tag{3.1.12}$$

So, the exponential transformation of $f_{ij}(t)$, is

$$\begin{aligned} f^e_{ij}(s) &= \int_0^\infty f_{ij}(t)e^{-st}dt \\ &= \sum_{\substack{r=1 \\ r\neq j}}^{n} p_{ir}\int_0^\infty\int_0^t h_{ir}(\sigma)f_{rj}(t-\sigma)e^{-st}d\sigma dt + p_{ij}\int_0^\infty h_{ij}(t)e^{-st}dt \\ &= \sum_{r=1}^{n} p_{ir}\int_0^\infty\int_0^t h_{ir}(\sigma)f_{rj}(t-\sigma)e^{-st}d\sigma dt - p_{ij}\int_0^\infty\int_0^t h_{ij}(\sigma)f_{jj}(t-\sigma)e^{-st}d\sigma dt \\ &\quad + p_{ij}\int_0^\infty h_{ij}(t)e^{-st}dt \\ &= \sum_{r=1}^{n} p_{ir}h^e_{ir}(s)f^e_{rj}(s) + p_{ij}h^e_{ij}(s)[1-f^e_{jj}(s)] \end{aligned} \tag{3.1.13}$$

Therefore, the mean first passage time for a process from state i to j is

$$\bar{\Theta}_{i,j} = \frac{d}{ds} f^{e}_{i,j}(s)\Big|_{s=0}$$

$$= \sum_{r=1}^{n} p_{i,r} \left[\frac{d}{ds} h^{e}_{i,r}(s) f^{e}_{r,j}(s) + h^{e}_{i,r}(s) \frac{d}{ds} f^{e}_{r,j}(s) \right]\Big|_{s=0} + p_{i,j} \frac{d}{ds} h^{e}_{i,j}(s)\Big|_{s=0}$$

$$- p_{i,j} \left[\frac{d}{ds} h^{e}_{i,j}(s) f^{e}_{j,j}(s) + h^{e}_{i,j}(s) \frac{d}{ds} f^{e}_{j,j}(s) \right]\Big|_{s=0}$$

Since $\frac{d}{ds} h^{e}_{i,j}(s)\Big|_{s=0}$ is the first moment of holding time distribution, i.e., the mean holding time $\bar{\tau}_{i,j}$, and

$$f^{e}_{i,j}(s)\Big|_{s=0} = \int_{0}^{\infty} f_{i,j}(t)dt = 1 ,$$

the mean first passage time $\bar{\Theta}_{i,j}$ can be derived as below.

$$\bar{\Theta}_{i,j} = \sum_{r=1}^{n} p_{i,r}(\bar{\tau}_{i,r} + \bar{\Theta}_{r,j}) - p_{i,j}\bar{\Theta}_{j,j} \qquad (3.1.14)$$

By Equation 3.1.2, Equation 3.1.14 can be written as :

$$\bar{\Theta}_{i,j} = \bar{\tau}_{i} + \sum_{r=1}^{n} p_{i,r}\bar{\Theta}_{r,j} - p_{i,j}\bar{\Theta}_{j,j} . \qquad (3.1.15)$$

However, the mean recurrence time $\bar{\Theta}_{j,j}$ is actually the reciprocal of the steady state entrance rate into state j, i.e.

$$\bar{\Theta}_{j,j} = \frac{1}{e_{j}} . \qquad (3.1.16)$$

Thus Equation 3.1.15 can be written as

$$\bar{\Theta}_{i,j} = \left[\bar{\tau}_i - \frac{p_{i,j}}{e_j}\right] + \sum_{r=1}^{n} p_{i,r}\bar{\Theta}_{r,j} \,. \tag{3.1.17}$$

and its matrix form is

$$\Theta = K + P\Theta \,, \tag{3.1.18}$$

where $\Theta = \left[\bar{\Theta}_{i,j}\right]$ and $K = (TU) - E^{-1}$ where T, U and E are :

(1) T is a diagonal matrix in which $T_{i,i} = \bar{\tau}_i$, and $T_{i,j} = 0$ otherwise,

(2) U is a unit matrix, i.e., all elements are one, and

(3) E is a diagonal matrix in which $E_{i,i} = e_i$, and $E_{i,j} = 0$ if $i \neq j$.

Therefore, the mean first passage time matrix Θ can be derived as:

$$\Theta = \frac{K}{I - P} \tag{3.1.19}$$

3.2. Model Behavior

In this section we use the measures that were defined previously to predict the key system characteristics for given stochastic transition probability matrix and holding time density function matrix which were estimated from the collected data. By solving the semi-Markov model we discover that the system makes a transition every 9 minutes and 8 seconds, on average. In comparing this with the mean time between error (MTBE) listed in Table 3.1, it is clear that most often the transitions are from one workload state to another. Also note that the model indicates an MTBE of 4152 hours for CPU errors. This number is estimated by solving the model equations although there were no observations in the measured period. (In examining the error data over a one year period we found two CPU

Table 3.1. Mean time between errors

Type of error	Frequency count	%	Mean time between errors (hour)
CPU	0	0	4152
CHAN	13	1.88	26.88
SWE	201	29.05	1.75
DASD	401	57.95	0.87
MULT	77	11.12	4.62

errors.) The table also shows that a disk error occurs (as indicated in the model) almost every 52 minutes while a software error is detected every 1 hour and 45 minutes. Most of the disk errors (95%) are recovered through hardware recovery (i.e., hardware instruction retry or ECC correction), thus resulting in negligible overhead. This shows that on-line recovery is highly effective and provides a system with the ability to tolerate a fault and recover almost instantaneously. Thus, a highly reliable system is achieved.

Table 3.2 lists the mean recurrence time for recovery routines. It shows that the on-line hardware recovery routine is invoked once every 0.62 hours, while a software recovery occurs every 2.57 hours. As mentioned earlier, hardware recovery involves hardware instruction retry or ECC correction. The maximum number of retries is predetermined. In the measured system each CPU has a 26-nanosecond machine cycle time and the disk seek time is about 25 milliseconds. We estimate a worst case hardware recovery cost of 0.5 seconds, i.e., incorporating twenty I/O retries: ten through the original I/O path and another ten through an alternative I/O path if the alternative is available. This, of course, overestimates the cost of hardware retry used for the CPU errors. However, the impact is very

Table 3.2. Mean recurrence time

Type of recovery	Mean recurrence time (hour)
Hardware	0.62
Software	2.57
Alternative	-
Off-line	651.37

insignificant. This can be seen by comparing the estimated time for each hardware recovery with the recovery overhead. The comparison shows that the cost of hardware recovery is worth only 0.02% of total computation time. The mean recurrence time of the alternative recovery routine, is not estimated due to lack of data, i.e., this event seldom occurred.

The model characteristics are summarized in Table 3.3. A dashed line in this table indicates a negligible value (less than 0.00001 probability). Table 3.3(a) shows the normal system behavior. Given that a transition has occurred the system CPU load is most likely to reach to W_7 or W_8, i.e. 0.72 or above. This is also reflected in the entry and occupancy probabilities (e and Φ). From the occupancy probabilities we see that almost 34% of the time the CPU load is as high as 0.96 (W_8); 39% of the time the CPU is moderately loaded (W_6 + W_7).

Table 3.3(b) shows the erroneous system behavior. The table indicates that about 30% of the transitions are to an error state (obtained by summing all the π's for all the error states). The DASD errors have the highest transition and entry probabilities. Since a transition occurs every 9 minutes, we estimate that an error is detected, on the average, every 30 minutes. Of course, over 98% of these errors caused negligible overhead.

Table 3.3. Summary of model characteristics

(a). CPU bound workload states

Measure	Workload state								
	W_0	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8
Φ	0	0.0625	0.0008	0.0136	0.1258	0.0054	0.1639	0.2255	0.3398
π	0.0257	0.0264	0.0014	0.0104	0.0559	0.0047	0.0635	0.1125	0.1127
e	0.00005	0.00005	-	0.00002	0.0001	0.00001	0.00012	0.00021	0.00021
$\overline{\Theta}$	5.78	5.62	102.56	14.32	2.65	31.38	2.33	1.32	1.32

(b). Error and recovery states

Measure	Error state					Recovery state			
	CPU	CHAN	SWE	DASD	MULT	HWR	SWR	ALT	OFFL
Φ	0	0.00005	0.0066	0.0383	0.0179	0.00022	0.00011	-	-
π	0.00004	0.0055	0.0850	0.1692	0.0322	0.2379	0.0572	0.00004	0.00023
e	-	0.00001	0.00016	0.00032	0.00006	0.00045	0.00011	-	-
$\overline{\Theta}$	4152	26.88	1.75	0.87	4.62	0.62	2.57	4089.5	651.57

An interesting characteristic of the multiple error events is also seen in Table 3.3(b). Although, the transition probability (π) of a MULT error is lower than that for SWE (0.0322 vs. 0.0850), its occupancy probability (Φ) is higher (0.0179 vs. 0.0066). This is due to the fact that a MULT error has a longer sojourn time as compared to SWE error events (293 seconds vs. 41 seconds from Table 2.4).

3.3. Effect of Workload

In this section we compute the steady state probability of being in a specified workload state and making a transition to a specified error state. Table 3.4 shows the

probabilities of a error occurring at various load levels. In this table, "time" refers to the mean holding time in the specified workload state (e.g., CPU = 0.96) before the process making a transition to the selected state (e.g., CHAN). An important relation between error probability and holding time in a workload state is seen in this table. The error probabilities appear to be not only a function of resource usage [10], but also related to the length of the holding time in a resource usage state. For example, in Table 3.4(a) the probability of a

Table 3.4. Holding time and transition probabilities to error states

(a). CPU workload

CPU	Error state								Total
	CHAN		SWE		DASD		MULT		
Load	Time	Prob	Time	Prob	Time	Prob	Time	Prob	Prob
0.96	668.18	0.0011	1609.71	0.0786	1218.62	0.1296	1641.20	0.0285	0.2377
0.72	596.28	0.0032	1118.12	0.0492	971.62	0.0990	757.09	0.0146	0.1661
0.54	1304.96	0.0010	1507.92	0.0471	1070.10	0.0489	722.26	0.0052	0.1027

Time - in seconds.

(b). I/O workload

DASD	Error state								Total
	CHAN		SWE		DASD		MULT		
Load	Time	Prob	Time	Prob	Time	Prob	Time	Prob	Prob
96.20	0	0	256.23	0.0022	434.54	0.0162	578.97	0.0046	0.023
67.34	898.35	0.0049	1243.35	0.0987	1170.84	0.1840	928.12	0.0262	0.3138
41.59	4522.67	0.0214	1516.92	0.0875	1148.18	0.1198	1286.67	0.0234	0.2521

Time - in seconds.

channel error is almost the same for two different CPU loads, 0.96 and 0.54. The mean holding time before a channel error occurs at the lower load is larger than that for the higher load, i.e. 1304.96 seconds versus 668.18 seconds. When the holding times are similar, however, (or increasing with increased usage), the error probabilities do increase with increasing resource usage. A similar phenomenon also exists for the I/O workload (see Table 3.4(b)). Thus, not only does a higher workload result in a higher error probability (for similar holding times), but the error probability also increases with increased holding time in a particular state. In other words, the error probability appears to be a function of the absolute amount of resource consumed in a given state, be it through increased workload and/or increased holding times. An explanation for this apparent "wear out" phenomenon is not clear (since a large majority of our errors are transient), but it certainly calls into further question the validity of the frequently used constant error probability assumption often made in reliability modeling.

3.4. Model Validation

In Chapter 2 we had shown that the resource-usage/error/recovery process of the measured system is best modeled as a semi-Markov process. This is due to the fact that the waiting and holding time distributions of some states are not exponentials. In order to validate this semi-Markov assumption we will compare the occupancy probabilities of states predicted from the model with the values estimated from the collected data.

From Equation 3.1.7 we know that the state occupancy probability of the model, Φ_j, is defined as :

$$\Phi_j = \frac{\pi_j \bar{\tau}_j}{\sum_{i=1}^{n} \pi_i \bar{\tau}_i} .$$

However the actual occupancy probability, denoted as $\hat{\Phi}_i$, can be estimated from the collected data by using the following equation.

$$\hat{\Phi}_i = \frac{\text{total time that system observed to be in state } i}{\text{length of the observation period}} . \qquad (3.4.1)$$

Table 3.5 lists the comparison of these two measures, Φ and $\hat{\Phi}$, for the normal states with significant occupancy probability (greater than 0.1 probability) and for one key error state (DASD). From Table 3.5 we see that the predicted probabilities closely match those estimated from the collected data (with the maximum of 0.025 tolerance, i.e., $\frac{\epsilon}{\hat{\Phi}}$). This

Table 3.5. Comparison of occupancy probabilities for different states

State	W_4	W_6	W_7	W_8	DASD
Φ	0.1258	0.1639	0.2255	0.3398	0.0383
$\hat{\Phi}$	0.1259	0.1634	0.2311	0.3452	0.0386
ϵ	0.0001	0.0005	0.0056	0.0054	0.0003
$\frac{\epsilon}{\hat{\Phi}}$	0.0008	0.0031	0.0242	0.0156	0.0078

Φ : predicted occupancy probability
$\hat{\Phi}$: actual occupancy probability
ϵ : the absolute error, $|\Phi - \hat{\Phi}|$

indicates that the semi-Markov model is a good model for the resource-usage/error/recovery process of the measured system.

3.5. Markov Versus Semi-Markov

In this section we investigate the significance of using a semi-Markov model to describe the overall resource-usage/error/recovery process. It has been argued that since errors only occur infrequently (i.e., λ is small), a Markov model may well approximate the real behavior. Thus, although the collected data shows that the semi-Markov process is a better model for the resource-usage/error/recovery process, i.e., more closely approximates the data from the measured system, it is reasonable to ask what deviations may occur if a Markov process is assumed. In order to answer this question we use a Markov model to describe the resource-usage/error/recovery process of the measured system and compare the results with those obtained through the more realistic semi-Markov model.

Two measures, the unconditional transition probability to the next state (γ) and the first passage time (Θ), are used as the basis for comparison.

3.5.1. Unconditional transition probability (γ)

Given a stochastic transition probability matrix $\left[p_{i,j}\right]$ and a holding time density function matrix $\left[h_{i,j}(t)\right]$, the unconditional transition probability from state i to state j, denoted as $\gamma_{i,j}$, in the semi-Markov process is given by [25]:

$$\gamma_{i,j} = \frac{\pi_i p_{i,j} \bar{\tau}_{i,j}}{\bar{\tau}}, \tag{3.5.1}$$

where $\bar{\tau}_{ij}$ is the mean holding time before a transition occurs from state i to state j and $\bar{\tau}$ is the mean holding time of the process. Because the selection of the next state in the Markov process is not dependent on the holding time in the current state, i.e. $\bar{\tau}_{ij} = \bar{\tau}_{ik}$ for every j and k, so from Equation 3.1.2 we can have

$$\bar{\tau}_i = \sum_{j=1}^{n} p_{ij}\bar{\tau}_{ij} = \bar{\tau}_{ij} \sum_{j=1}^{n} p_{ij} = \bar{\tau}_{ij} \cdot \tag{3.5.2}$$

Substitute this into Equation 3.5.1 we have

$$\gamma_{ij} = \frac{\pi_i p_{ij} \bar{\tau}_i}{\bar{\tau}}. \tag{3.5.3}$$

Further, from Equation 3.1.7 we have

$$\gamma_{ij} = \Phi_i p_{ij}. \tag{3.5.4}$$

Table 3.6 compares the unconditional transition probability for semi-Markov and Markov models. We see from Table 3.6(a) that when the CPU load is as high as 0.96, the transition probabilities to the software and multiple errors are close for both models. This is also true for channel error when the CPU load is 0.54. This is because for some destinations j the holding time to the next state is the same as the waiting time of the current state, i.e., $\bar{\tau}_i = \bar{\tau}_{ij}$. For the majority of the cases, however, the Markov and semi-Markov models are not in agreement. Table 3.7 shows the ratios of the unconditional transition probability γ_{ij} estimated from both models, Markov versus semi-Markov. If the ratio is less then 1 then the Markov process underestimates the transition probability, otherwise, it overestimates. From this table we see that the Markov assumption sometimes overestimates and sometimes underestimates the transition probability. In particular it overesti-

Table 3.6. Comparison of transition probabilities, $\gamma_{i,j}$
(Markov versus Semi-Markov)

(a). From CPU workload to error states

Load	Model	Error			
		CHAN	SWE	DASD	MULT
0.96	semi-Markov	0.0011	0.0786	0.1296	0.0285
	Markov	0.0025	0.0783	0.1705	0.0278
0.72	semi-Markov	0.0032	0.0492	0.0990	0.0146
	Markov	0.0058	0.0469	0.1086	0.0206
0.54	semi-Markov	0.0010	0.0471	0.0489	0.0052
	Markov	0.0011	0.0429	0.0627	0.0099

(b). From I/O workload to erro states

Load	Model	Error			
		CHAN	SWE	DASD	MULT
96.20	semi-Markov	0	0.0022	0.0162	0.0046
	Markov	0	0.0082	0.0350	0.0075
67.34	semi-Markov	0.1840	0.0987	0.0263	0.0049
	Markov	0.0068	0.0987	0.1955	0.0352
41.59	semi-Markov	0.0214	0.1198	0.0875	0.0234
	Markov	0.0069	0.0839	0.1516	0.0264

Table 3.7. Ratio of $\gamma_{i,j}$ (Markov/semi-Markov)

Workload		Error			
Resource	Load	CHAN	SWE	DASD	MULT
CPU	0.96	2.273	0.996	1.316	0.976
	0.72	1.818	0.953	1.097	1.411
	0.54	1.100	0.911	1.282	1.904
DASD	96.20	-	3.727	2.161	1.631
	67.34	0.037	1.0	7.446	7.184
	41.59	0.322	0.700	1.733	1.128

mates the transition probabilities to most error states, regardless of the state. Overestimation will lead to an unduly conservative reliability estimate and underestimation to an overly optimistic estimate. Thus both are undesirable.

3.5.2. First Passage Time (Θ)

We now examine the difference between the first passage times under the Markov and the semi Markov assumptions. The first passage time distribution can be used to estimate the MTBE and its variance.

The mean first passage time from state i to state j, $\overline{\Theta}_{i,j}$ in a semi-Markov process is given in Section 3.1 as :

$$\overline{\Theta}_{i,j} = \begin{cases} \dfrac{1}{e_j} & \text{if } i = j \\ \overline{\tau}_i + \sum_{\substack{r=1 \\ r \neq j}}^{n} p_{i,r}\overline{\Theta}_{r,j} & \text{otherwise} \end{cases} \qquad (3.5.5)$$

From this equation, we notice that the mean first passage time depends on only the mean holding time and the conditional transition probability of the current state. Clearly, if the first moment of the first passage time to the error state (e.g., the MTBE) is the only main concern, the Markov process should be able to provide adequate information. If the distribution (or the higher moments) of the first passage time is of interest, the Markov model may be inadequate, particularly if the variance of the first passage time is large. This can be seen clearly from the following equation [25].

$$\overline{\Theta^2}_{i,j} = \begin{cases} \frac{1}{\pi_j}\left(\sum_{i=1}^{n} \pi_i \overline{\tau^2}_i + \sum_{\substack{r=1 \\ r \neq j}}^{n} \sum_{i=1}^{n} 2\pi_i p_{i,r} \overline{\tau}_{i,r} \overline{\Theta}_{r,j}\right) & \text{if } i = j \\ \overline{\tau^2}_i + \sum_{\substack{r=1 \\ r \neq j}}^{n} p_{i,r}\left(2\overline{\tau}_{i,r}\overline{\Theta}_{r,j} + \overline{\Theta^2}_{r,j}\right) & \text{otherwise} \end{cases} \qquad (3.5.6)$$

This equation indicates that the second moment of the first passage time is a function of the second moment of the state waiting time, as well as the mean holding time to the next state. Since the mean ($\overline{X}$) and the standard deviation (σ) of an exponential distribution are the same, and the second moment of an exponential distribution is only a function of its mean, i.e. $E[X^2] = 2*E[X]^2$. Thus, a Markov assumption may under- or over-estimate the second moment, $E[X^2]$, if $\sigma \neq E[X]$.

Table 3.8 shows the ratio of $\overline{\Theta^2}_{i,j}$ (Markov/semi-Markov) for transitions from a few selected workload states to the error states. From Table 3.8, we see that the Markov assumption frequently underestimates the second moment of the first passage time (to the error state). The underestimation can be as much as 30%. However, it overestimates the variation of first passage time among different resource usage states, although this is not

Table 3.8. Ratio of $\overline{\Theta^2}$ (Markov/semi-Markov)

Workload		Error			
Resource	Load	CHAN	SWE	DASD	MULT
CPU	0.96	0.989	0.876	0.694	0.939
	0.72	0.996	0.998	0.909	0.972
	0.54	0.992	0.937	0.830	0.953
DASD	96.20	1.006	1.010	0.901	0.991
	67.34	1.002	0.963	0.888	0.972
	41.59	1.001	0.948	0.871	0.963

shown in the table.

3.5.3. Summary

In summary, our measurements show that using a Markov model frequently overestimates the unconditional transition probabilities and underestimates the variance of the first passage times to the error states. The overestimation, of course, will lead to an unduly conservative reliability prediction. It can be argued that such gross overestimation (as seen in some cases here) is undesirable and may not be cost beneficial. The underestimation is no doubt a serious problem which may lead to unduly optimistic reliability prediction.

CHAPTER 4

SOFTWARE RELIABILITY MODEL

4.1. Introduction

The problem of modeling software reliability during the development, debugging and validation phases of the software cycle is a well researched area. However, there are few studies which model software error and recovery processes in a fully operational production environment. The difficulties are partly due to the fact that, unlike computer hardware, which is reasonably modularized, each software system can have its own peculiar characteristics. At this stage, it is extremely valuable to develop a comprehensive model quantifying the software error and recovery processes in a production system using real data. In addition to providing useful information on how and when errors occur in the real world, this process provides the quantification of the interaction among different types of software errors; an important result for developing analytical models.

In this chapter, a state-transition model to describe the software error and recovery processes in a complex operating system is described. Measurements were made on an MVS (Multiple Virtual Storage) system running on an IBM 3081 mainframe. Time-stamped low level error and recovery data from MVS, collected during the normal operation of the system, formed the basis for developing the model. The semi-Markov model developed from the real data provides a quantification of the system error characteristics and also gives an

insight into the interaction between the various software error and recovery processes occurring during normal system operation.

4.1.1. Related Research

Most software reliability models usually refer to the development, debugging and testing phases of the software [11, 12, 13, 14]. Few of these models have been applied to the operational phase of the software. In [2] and [15], software failures in an operating environment are studied. Both studies found that at least 60% of system failures were software related. There has been little explicit study of hardware/software reliability. In [15], software failures related to hardware problems in the operating system are analyzed and it is shown that errors in the hardware/software interface are often fatal. In [27], a resource-usage/reliability model was developed from real data and it was seen that about 36% of detected errors (not necessarily system failures) were related to software problems.

With the exception of software reliability growth models, which have been validated with real data, there are few, if any, models of software reliability in an operational environment. Exceptions are the hardware and software model discussed in [18] and a measurement-based model of workload dependent failures discussed in [10]. However, these only describe the external behavior of the system and do not provide insight into component-level behavior.

It is therefore highly instructive to develop a detailed model based on low-level error data from a production system. In the following sections, we construct a error/recovery model for the MVS operating system. Software problems of differing severity are modeled. Multiple errors are also considered and and the effect of on-line recovery routines is taken into account.

4.2. Error Characterization

In this section the collection and characterization of the software error and error recovery data are discussed. A state-transition diagram is developed to describe the different error and recovery states. This allows us to determine the serverity of errors and effectiveness of recovery.

Error data based on different causes were collected. Information on software errors is automatically logged by an operating system module. Details of the logging mechanism are described in [23]. In order to ensure that the analysis is not biased by error records relating to the same problem, two levels of data reduction which were described in Chapter 3 were performed. As a result, the software errors were classified into eight classes. These eight classes are called error events, since they may contain more than one error, and are defined as follows.

(1) Control (CTRL)	- incidents indicating the invalid use of control statements and invalid supervisor calls
(2) Deadlocks (DLCK)	- incidents indicating system or operator detected endless loop, endless wait state or violation of system or user-defined time limits
(3) I/O and Data Management (I/O)	- incidents indicating problems occurred during I/O management or during the creation and processing of data sets
(4) Storage Management (SM)	- incidents indicating errors in the storage allocation/de-allocation process or in virtual memory mapping
(5) Storage Exceptions (SE)	- incidents indicating addressing of nonexistent or inaccessible memory locations
(6) Programming Exceptions (PE)	- incidents indicating program errors other than storage exceptions

(7) Others (OTHR) - incidents indicating that problems occurred which do not fit the above categories

(8) Multiple Errors (MULT) - incidents indicating more than one type of error listed above

Table 4.1 lists the frequencies of different types of software error events defined above. The table shows that more than one half (52.5%) of software errors were I/O and data management errors and another 11.4% of the errors were storage management errors. A significant percentage (17.4%) of errors were classified as multiple errors and are specifically modeled in the following sub-section.

4.2.1. Multiple Errors

A multiple error most often is due to a single fault that has non-identical manifestations provoked by different types of system activity. Since the manifestations are not

Table 4.1. Frequency of software errors

Type of Errors	Frequency	Percent
Control	213	7.72
Deadlock	23	0.84
I/O & Data Management	1448	52.50
Program Exception	65	2.43
Storage Exception	149	5.40
Storage Management	313	11.35
Others	66	2.32
Multiple Error	481	17.44
Total	2758	100.00

identical, recovery may be complex. Figure 4.1 shows the state-transition diagram of a multiple error developed from the data. The transition probability from state i to state j, p_{ij}, is estimated from the measured data using:

$$p_{ij} = \frac{\text{observed number of transitions from state } i \text{ to state } j}{\text{observed number of transitions from state } i}.$$

This figure not only illustrates the possible interactions among different software errors but also provides detailed information on the occurrence of transitions. For example, if a program exception error (PE) occurs, there is about a 63% chance that a storage exception (SE) on error will follow. Further, there is more than a 50% chance that one storage error will be followed by another error of the same type (52% for storage management and also for storage exception). If we only focus on those transitions with significant probabilities (i.e., higher than 0.1), the number of states in Figure 4.1 can be reduced to five. The state-transition diagram for these active states is illustrated in Figure 4.2. Notice that a cyclic path is formed by the I/O and data management (I/O) along with the two different types of exception states (program exception and storage exception).

4.2.2. Recovery Modeling

Recovery in MVS is designed as a means by which the system can prevent a total loss. Whenever a program is abnormally interrupted due to the detection of an error, the Supervisor gets control. If the problem is such that further processing could degrade the system or destroy data, the Supervisor gives control to the Recovery Termination Manager (RTM). If a recovery routine is available for the problem program, RTM gives control to this routine before deciding to terminate the program.

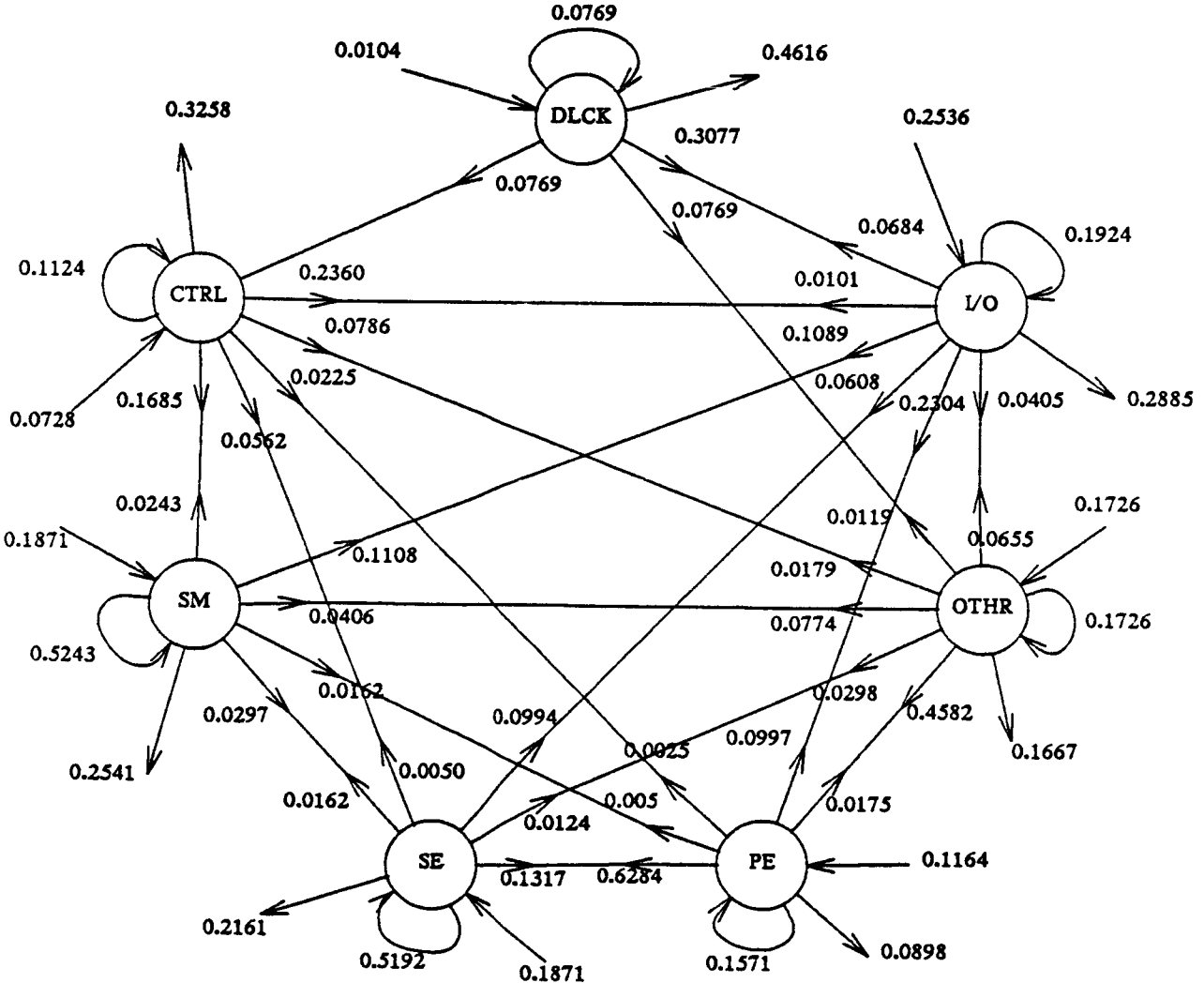


Figure 4.1. State-transition diagram for a multiple error

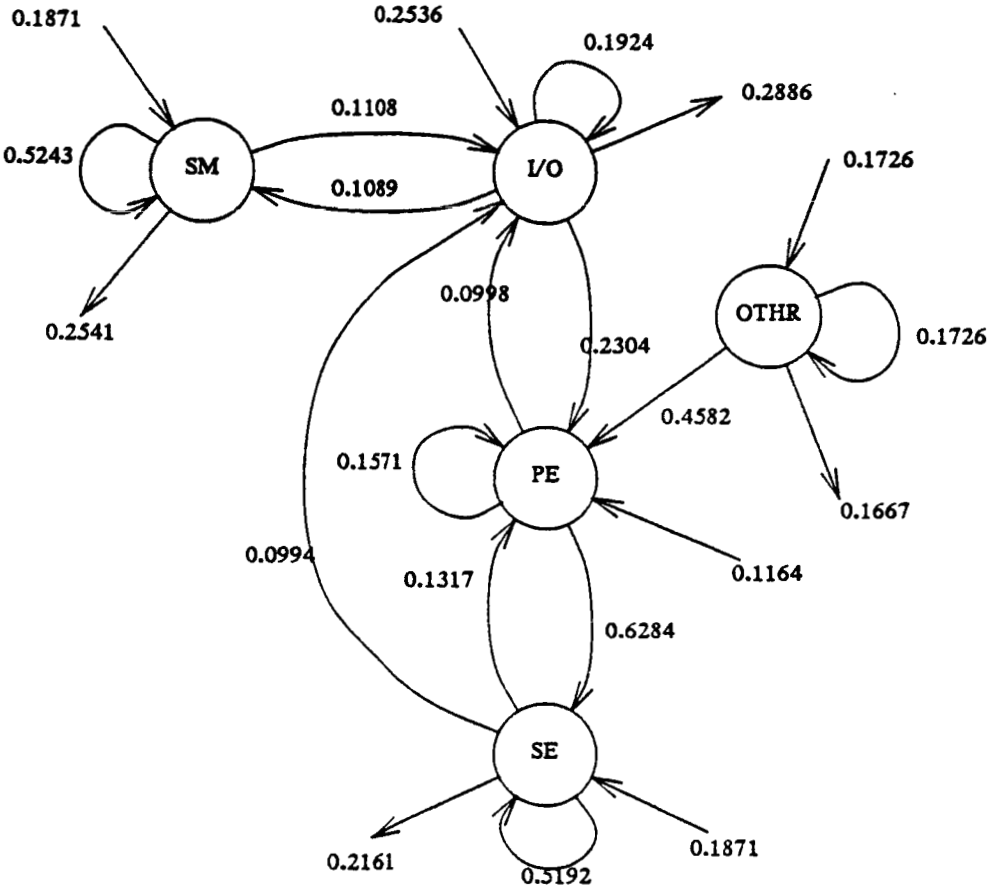


Figure 4.2. Reduced state-transition diagram of multiple errors

The purpose of a recovery routine is to free the resources kept by the failing program (if any), to locate the error, and to request either a continuation of the termination process or a retry. Recovery routines are generally provided to cover critical MVS functions. It is

however, the responsibility of the installation (or of the user) to write a recovery routine for other programs.

More than one recovery routine can be specified for the same program: if the latest recovery routine asks for a termination of the program, the RTM can give control to another recovery (if provided). This process is called "percolation." The percolation process ends if either a routine issues a valid retry request or no more routines are available. In the latter case, the program and its related subtasks are terminated. If a valid retry is requested, a retry routine restores a valid status using the information supplied by the recovery routine(s) and gives control to the program. In order for a retry to be valid, the system should verify that there is no risk of error-recurrence and that the retry address is properly specified. An error may have four possible effects.

1) Retry - The system successfully recovered and returned control to the problem program.

2) Task Termination - The program and its related subtasks are terminated, but the system is not affected.

3) Job Termination - The job in control at the time of the error is aborted.

4) System Damage - The job or task in control at the time of the error was critical for system continuation. Thus, job/task termination resulted in system failure.

Figure 4.3 illustrates the steps in the recovery process. It is clear that recovery can be as simple as a retry or more complex, requiring several percolations before a retry. The problem can also be such that no retry or percolation is possible. Table 4.2 shows the percentage for these different types of situations. For example, for storage management errors, approximately 8% of the cases resulted in a direct retry, 84% involved some percolation and over

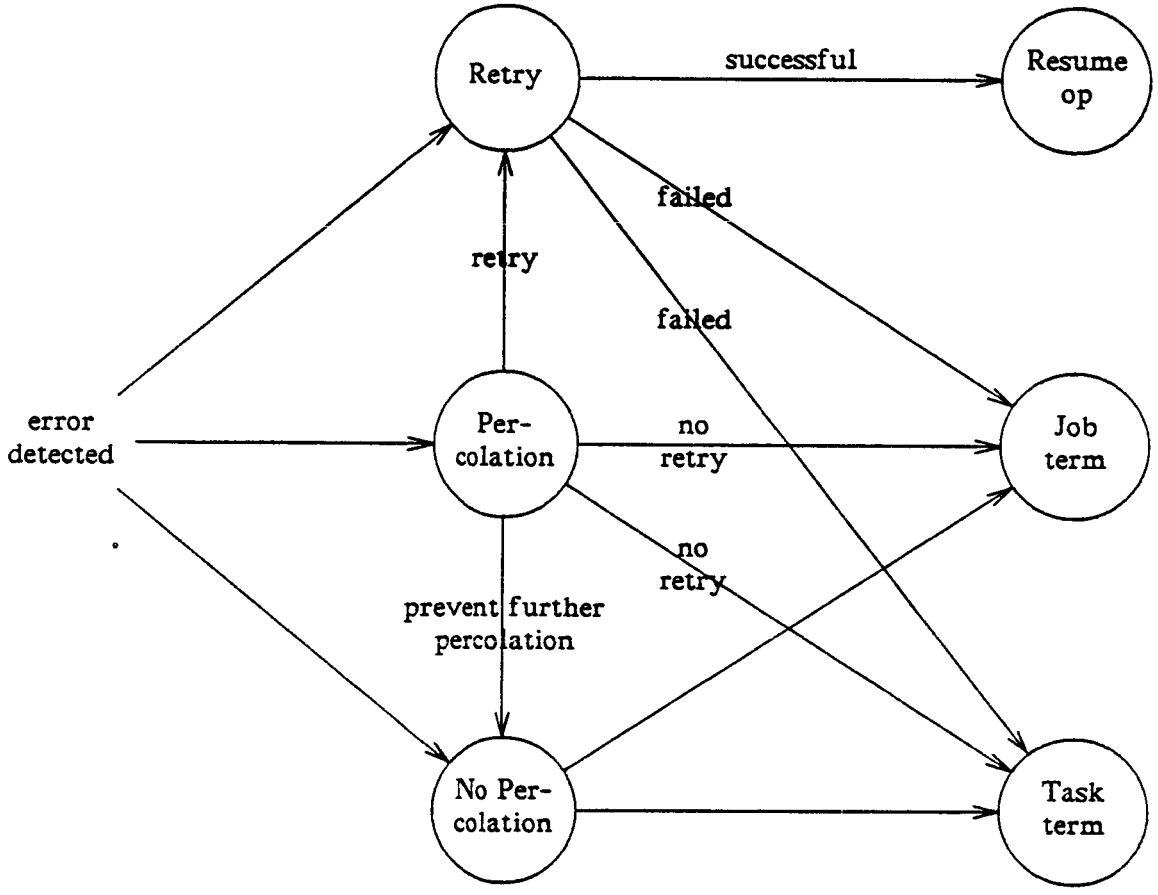


Figure 4.3. Flow of recovery

Table 4.2. Percentages of recovery attempts for a software error

Type of error	Retry (%)	Percolation (%)	No-Percolation (%)
Control	78.38	21.62	0.0
Deadlock	2.78	97.22	0.0
I/O & Data Management	93.49	6.51	0.0
Program Exception	20.09	79.91	0.0
Storage Exception	28.09	71.91	0.0
Storage Management	7.77	83.73	8.50
Others	14.89	85.11	0.0

8% could not be percolated any further (i.e. jobs/task termination). The table shows that only in a small percentage of the cases was the problem un-recoverable (no-percolation).

4.3. Software Reliability Model

4.3.1. Overall Error/Recovery Model

In this section we combine the separate error and recovery models to construct a single overall model shown in Figure 4.4. Note that a state, Normal, represents the normal system operation. The results of the recovery process are classified into three different states (resume op, task term and job term) to reflect the severity of errors. The model thus provides a complete overview of software error and recovery from an error occurrence to its recovery.

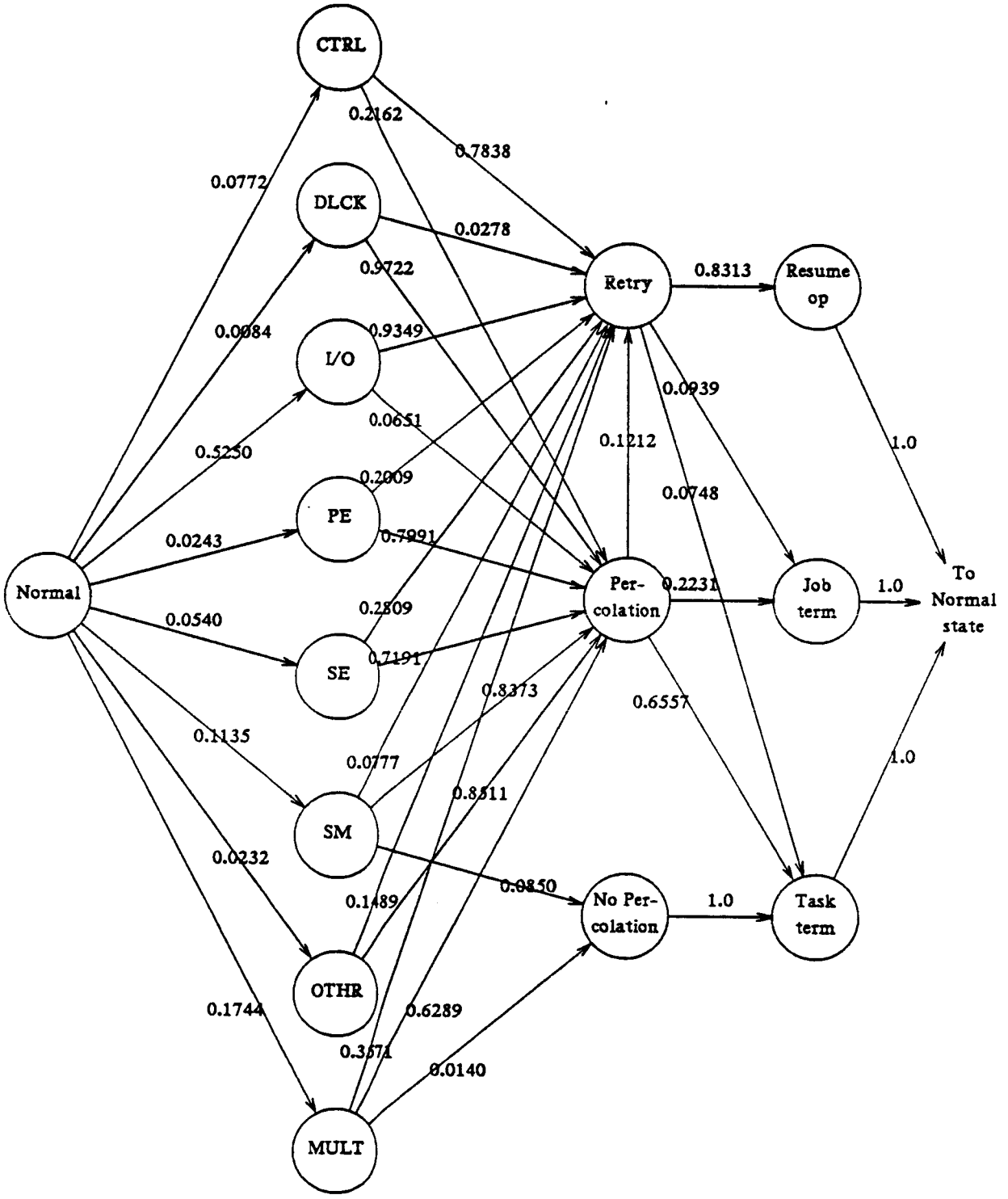


Figure 4.4. Software error/recovery model

4.3.2. Waiting Time Distributions

Table 4.3 shows the characteristics of both normal and error states in terms of their waiting times. Note that the duration of a single error is generally in the range of 20-40 seconds on the average, except for deadlock and "others". The table also shows that the errors not classified are relatively insignificant since their duration is less than 2 seconds. Program exceptions take twice as long as control errors (42 seconds versus 21 seconds). This is possibly due to the extensive software involvement in recovering from program exceptions. Figure 4.5 shows the density of waiting time in the normal operation state, i.e., the density of the time to error. This density could not be fitted to a simple exponential, and because of the shape of this density we found that it was fitted to a multi-stage gamma

Table 4.3. Mean waiting time (in seconds) of states

State	# of obs	Mean waiting time	Standard deviation	Std Error of mean
Normal	2757	10461.33	32735.04	623.44
Control	213	21.92	84.21	5.77
Deadlock	23	4.72	22.61	4.72
I/O & Data Management	1448	25.05	77.62	2.04
Program Exception	65	42.23	92.98	11.53
Storage Exception	149	36.82	79.59	6.52
Storage Management	313	33.40	95.01	5.37
Others	66	1.86	12.98	1.60
Multiple Error	481	175.59	252.79	11.53

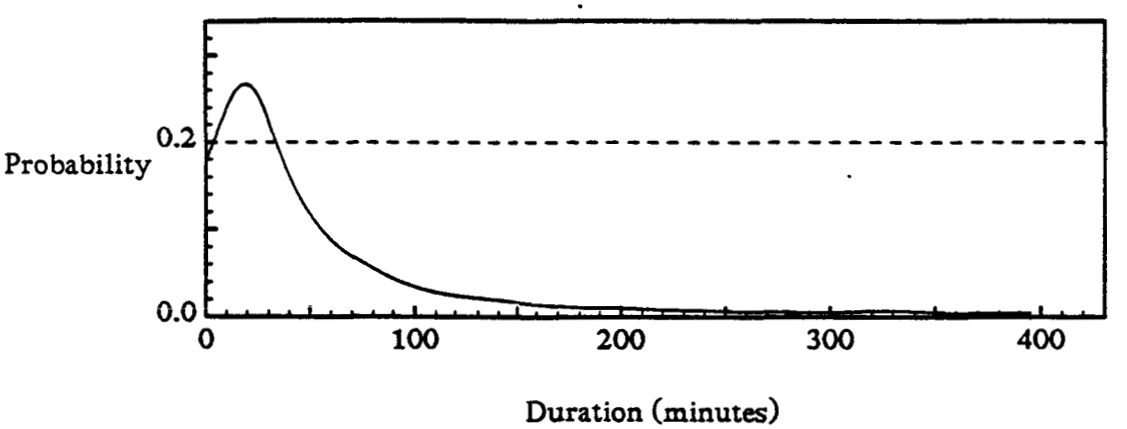


Figure 4.5. Time to error density

density function better than to a phase-type exponentials at the same acceptable level. The multi-stage gamma density function $f(t)$ is defined as

$$f(t) = \sum_{i=1}^{n} a_i g(t; \alpha_i, s_i), \tag{4.3.1}$$

where $a_i \geq 0$, $\sum_{i=1}^{n} a_i = 1$, and n is the number of stages. The $g(t; \alpha, s)$ is a gamma density function (with s the distance shifting from the origin),

$$g(t; \alpha, s) = \begin{cases} 0 & t < s \\ \dfrac{1}{\Gamma(\alpha)}(t-s)^{\alpha-1} e^{-(t-s)} & t \geq s \end{cases}, \tag{4.3.2}$$

where $\Gamma(\alpha)$ is a gamma function. Hence, the density in Figure 5 so obtained has five stages, given by

$$f(x) = 0.748\ g(x;\ 2.1,\ -1) + 0.055\ g(x;\ 0.5,\ 0) + 0.069\ g(x;\ 3.5,\ 3)$$
$$+\ 0.030\ g(x;\ 5.0,\ 8) + 0.098\ g(x;\ 5.0,\ 17),$$

tested using the Kolmogorov-Smirnov test [26] at the 0.01 significance level.

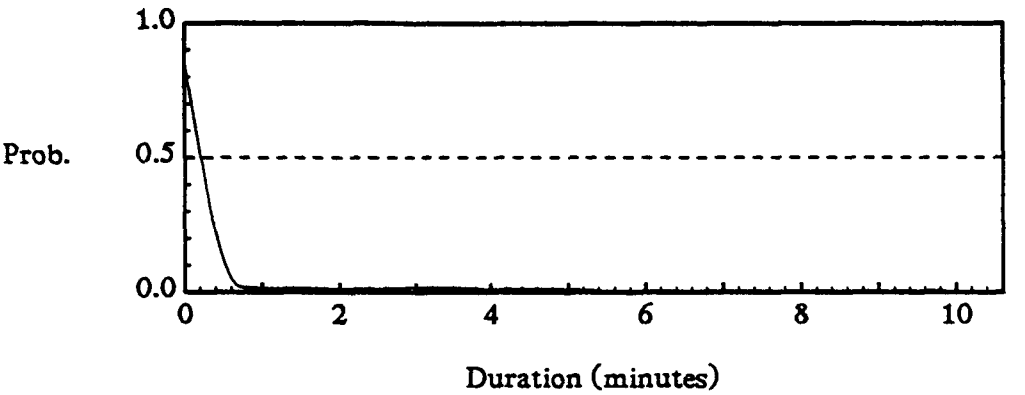
4.3.3. Recovery Time Distribution

For the purposes of evaluating the time for recovery, we assumed that each recovery mode takes a constant amount of time. The overall recovery time, i.e., the duration of an error event (or the waiting time in an error state), however was not constant, since an error event can involve more than one recovery attempt or may require more than one recovery routine. The recovery time was then computed as the time difference between the first and the last detected error caused by the same event. The duration of an error event was used to measure the effectiveness of recovery from this event and also the severity of the error.

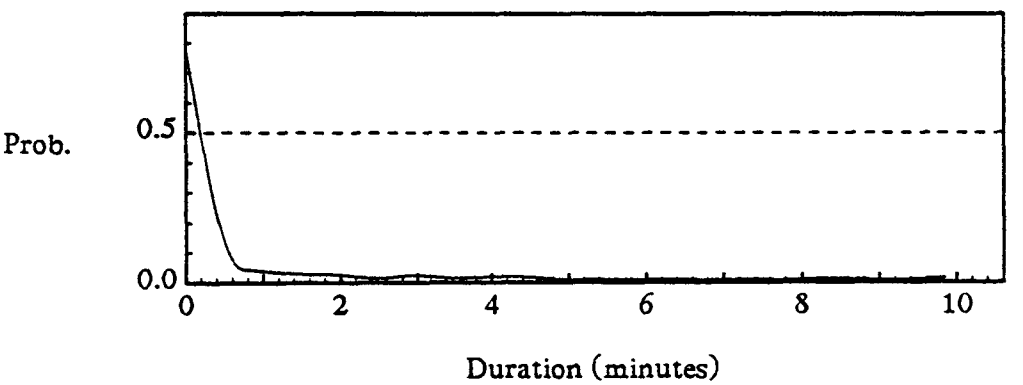
Figure 4.6 shows the recovery time densities for three different types of errors: I/O and data management, storage management, and multiple errors. Note that none of these densities could be fitted by simple exponentials at an acceptable level of significance. Thus, they were fitted to phase-type exponential density functions [26].

$$f(t) = \sum_{i=1}^{n} a_i g_i(t),$$

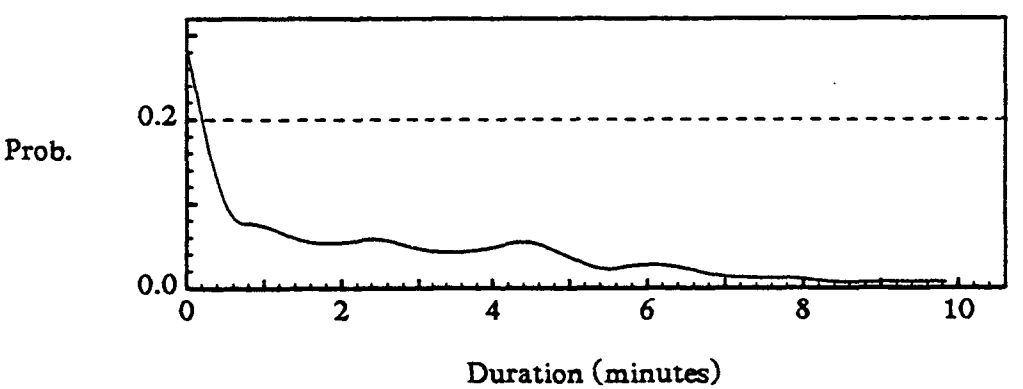
where $a_i \geq 0$, $\sum_{i=1}^{n} a_i = 1$ and n is the number of phases. The g_i function can be a simple exponential, a multi-stage hyperexponential, or a muti-stage hypoexponential density function. These exponentials are defined in Section 2.4. The densities in Figure 4.6 were fitted to the following functions (tested using the Kolmogorov-Smirnov test at the 0.01 significance level):



(a). Density for I/O and Data Management state



(b). Density for Storage Management State



(c). Density for multiple error state

Figure 4.6. Recovery time (error duration) densities

(1) I/O: $f(t) = 0.07825e^{-0.08594t} + 0.000354e^{-0.003958t}$,

(2) SM: $f(t) = 0.1642e^{-0.5t} + 0.030424e^{-0.053294t} + 0.0006634e^{-0.006586t}$, and

(3) MULT: $f(t) = 0.078e^{-0.4t} + 0.002426e^{-0.005866t} + 0.002163e^{-0.005525t}$.

4.3.4. Summary

In summary, the model developed explicitly quantifies the error and recovery process in the measured software. We note both the time to error and the recovery time distributions in several key states cannot be modeled as simple exponentials. Hence the overall process is modeled as a semi-Markov process. Further, the semi-Markov process is irreducible with the resume operation (resume op) state, the job termination (job term) state, and the task termination (task term) state being recurrent.

In the next sub-section we analyze the overall model to determine key software error characteristics. The mean time between different types of errors is evaluated along with model characteristics such as the occupancy probability of key error states.

4.4. Model Analysis

4.4.1. General Characteristics

By solving the semi-Markov model, we discover that the measured software system made a transition, on the average, every 43 minutes and 22 seconds. Table 4.4 lists the mean time between different software errors (i.e., mean time between errors) and Table 4.5 shows the mean recurrence time for recovery processes. By examining the mean recurrence time for I/O and MULT errors from Table 4.4 and comparing them with the mean waiting times in Table 4.3, we find that although the I/O errors occur about 3 times as often as the

Table 4.4. Mean time between errors

Error state	Frequency count	Percent (%)	MTBE (hour)
CTRL	213	7.72	37.83
DLCK	23	0.84	351.58
I/O	1448	52.50	5.56
PE	65	2.43	120.15
SE	149	5.40	54.08
SM	313	11.35	25.73
OTHR	66	2.32	125.84
MULT	481	17.44	16.75

Table 4.5. Mean recurrence time of recovery

Type of recovery	MTBR (hour)
Retry	4.25
Percolation	8.55
No-Percolation	241.43

multiple errors, the system spends nearly 6 times longer in recovering from a multiple error (25 seconds for I/O errors versus 175.6 seconds for multiple errors). This is because recovery from a multiple error involves several different types of recovery attempts. In addition, 63% of the multiple errors invoke percolation compared with the fact that 94% of the I/O errors recovered through retry (see Figure 4.4).

4.4.2. Model Probabilities

Given the irreducible semi-Markov model of Figure 4.4, the following steady state probabilities were evaluated. The derivations of these measures are given in Section 3.1.

(1) transition probability (π_j) - given that the process is now making a transition, the probability that the transition is to state j

(2) occupancy probability (Φ_j) - at any instant time the probability that the process occupies state j

(3) entry probability (e_j) - at any instant given that the process is entering a state, the probability that the process enters state j

(4) mean recurrence time ($\overline{\Theta}_j$) - mean recurrence time of state j

The model characteristics are summarized in Table 4.6. A dashed line in this table

Table 4.6. Characteristics of software error/recovery model

(a)

Measure	Normal state	Error state							
		CTRL	DLCK	I/O	PE	SE	SM	OTHR	MULT
π	0.2474	0.0191	0.0020	0.1299	0.0060	0.0134	0.0281	0.0057	0.0431
Φ	0.9950	0.00016	-	0.00125	0.000098	0.000189	0.00036	-	0.002913

(b)

Measure	Recovery state			Result		
	Retry	Percolation	No-Percolation	Resume op	Task term	Job term
π	0.1704	0.0845	0.0030	0.1414	0.0712	0.0348
$\overline{\Theta}^*$	4.25	8.55	241.43	5.11	10.16	20.74

* - in hour

indicates a negligible value (less than 0.00001 probability). From the occupancy probability, Φ, of the normal state in Table 4.6(a), we see that in about 99.5% of time the software system is operating normally, i.e., only 0.5% of time the system detects software errors. This indicates that the reliability of the measured software system can be as high as 0.995. In Section 2.2 we know that about 35% of observed errors were software errors. Thus the effect on the overall system reliability due to the software errors is very significant.

The table also shows that, of all possible transitions made, 24.73% are to an error state (obtained by summing all the π's for all the error states) and another 25.79% are to a recovery state. Since it was seen earlier that a transition occurs every 43 minutes, we estimate that a software error is detected, on the average, every 3 hours. From Table 4.6(b), we notice that although an error is detected almost every 3 hours, a successful recovery (i.e., results in resume operation), only occurs once every five hours, i.e., nearly 43% of the errors result in task/job termination.

Multiple-error events formed a significant category on their own. Since this type of event involves several errors and result in considerable overhead, it is analyzed separately in the next section.

4.4.3. Characteristics of A Multiple Error

In Section 4.2 we pointed out that about 17% of software errors were multiple errors. We also noticed that the multiple errors mostly consist of I/O, storage, or program errors. A strong connection between program and storage exception was seen in the occurrence of a multiple error. Table 4.7 lists the characteristics for a multiple error and was obtained by solving the semi-Markov model described in Figure 4.1 with a zero holding time in the normal state (i.e., given a multiple error occurs). From Table 4.7 we see (from π,

Table 4.7. Characteristics of a multiple error

Measure	Normal state	Error state						
		CTRL	DLCK	I/O	PE	SE	SM	OTHR
π	0.1767	0.0327	0.0048	0.1451	0.1473	0.2957	0.1360	0.0617
Φ	0	0.0648	0.0130	0.3004	0.0837	0.2202	0.2717	0.0462
e	0.00568	0.00105	0.00015	0.00466	0.00473	0.00950	0.00437	0.00198
$\overline{\Theta}^{\bullet}$	0.0489	0.2647	1.8126	0.0596	0.0587	0.0292	0.0636	0.1401

$\bullet$ - in hour

transition probability) that nearly 30% of the transitions are made to the storage exception state when the process enters a multiple error mode. Once in a multiple error mode, a storage exception error occurs every 1 minute and 45 seconds ($\overline{\Theta}$ = 0.0292 hours in Table 4.7), while the average duration of multiple errors is about 2 minutes and 56 seconds ($\overline{\Theta}$ = 0.0489 hours, the recurrence time of the normal state). Note that the average duration of a multiple error predicted here from the model is very close to the mean duration of a multiple error, 175.5 seconds obtained from real data, listed in Table 4.3. This provides a strong evidence that the semi-Markov process is a good model for our measured system due to its fairly accurate prediction. As soon as an entry into a multiple error is made, consecutive errors are detected almost every 31 seconds (by taking the reciprocal of the sum of all entry probabilities e in Table 4.7). This indicates that about 5 to 6 errors will be detected on average, once a multiple error occurs.

There are several interesting characteristics of multiple errors which can be derived from the model of Figure 4.1. For example, if we want to know the probability of a storage exception error given an I/O error, we can evaluate it by the multi-step transition

probability to the SE state from the I/O state. This turns out to be very small, only 0.0076. However, we find that the probability of an I/O occurring given a SE occurs at any time instant, is as high as 0.668. This is partly due to the fact that for a semi-Markov process the unconditional transition probability at any time instant, γ_{ij}, is not only a function of conditional transition probability p_{ij} but also a function of mean holding time. This can be seen in Equation 2.5.1.

4.5. Conclusion

In this study, we have developed a semi-Markov model to describe the error and recovery processes in the MVS system. The model is based on real error data collected during normal system operation. The semi-Markov model developed provides a quantification of system error characteristics and the interaction between different types of errors. As an example, we provide a detailed model and analysis of multiple errors, which constitute approximately 17% of all software errors and result in considerable overhead. It is suggested that other systems be similarly analyzed and modeled so that a wide range of realistic models of software reliability in an operating environment are available.

CHAPTER 5

PERFORMABILITY MODEL

A workload/reliability model is built based on real data. Given a stochastic transition probability matrix and a holding time density matrix, the system behavior such as the unconditional transition probability and state occupancy probability in the steady state can be estimated. However, the performability of the measured system is not yet addressed. Thus in this chapter we use the resource-usage/error/recovery model to estimate the performability of the system. Reward functions are used to depict the performance degradation due to errors and also due to different types of recovery procedures. Toward this end, we define a reward rate for each state of the resource-usage/error/recovery model.

5.1. Reward Function

First, we propose the reward rate r_i (per unit time) for each state i in our model as follow:

$$r_i = \begin{cases} \dfrac{s_i}{s_i + e_i} & \text{if } i \in S_N \bigcup S_E \\ 0 & \text{if } i \in S_R \ . \end{cases} \tag{5.1.1}$$

where, the s_i and e_i are the service rate and the error rate in state i, respectively. Thus one

unit of reward is given for each unit of time when the process stays in the normal states S_N. The penalty paid depends on the number of errors generated by an error event. With an increasing number of errors the penalty per unit time increases, and accordingly, the reward rate decreases. Zero reward is assigned to recovery states. This is due to the fact that during the recovery process the system does not contribute any useful work toward the system performance besides recovering from an error. Based on this proposal, reward rates for the error states are estimated and shown in Table 5.1. We know that from Table 3.3(b) the transition probability to the DASD error is about as much as twice to the SWE error and Table 5.1 shows that the reward gained from the DASD state is also as much as twice from the SWE state. Thus we expect that the impact due to the DASD error on the performability is much higher than that due to the SWE error. In order to understand the effectiveness of various errors, first we show some important performability measures that can be derived from the model.

Since the system can be in any state at any instant, so the reward rate of the system at time t, $X(t)$, is the reward rate of the state where the system is currently occupied. It is a random variable and denoted as

Table 5.1. Reward rates, r_i, for error states

State	DASD	SWE	CHAN	MULT
r_i	0.5708	0.2736	0.9946	0.2777

$$X(t) = \left\{ r_i \mid \text{process is in state } i \text{ at time } t \right\}. \tag{5.1.2}$$

Therefore the expected reward rate at time t, $E[X(t)]$, can be evaluated as

$$E[X(t)] = \sum_i p_i(t) r_i \ , \tag{5.1.3}$$

where $p_i(t)$ is the probability of the process being in state i at time t. The cumulative reward by time t, $Y(t)$, can be derived from

$$Y(t) = \int_0^t X(\sigma) d\sigma \ . \tag{5.1.4}$$

Therefore, the expected cumulative reward at time t, $E[Y(t)]$, is given by [28]:

$$E[Y(t)] = E\left[\int_0^t X(\sigma) d\sigma\right] = \sum_i r_i \int_0^t p_i(\sigma) d\sigma \ . \tag{5.1.5}$$

In order to solve for $p_i(t)$ and hence other measures, we convert the semi-Markov process into a Markov chain using the method of stages [26,29]. The conversion of the semi-Markov model to the Markov model for the measured system is described in section 5.2. Thus the state probability vector $P(t) = (....,p_i(t),....)$ can be computed by solving the set of differential equations of the form:

$$\frac{d}{dt} P(t) = P(t)Q$$

where Q is transition rate matrix of the Markov chain [25].

5.2. Semi-Markov to Markov Conversion

As we know, the sojourn time distribution of states is the only difference between semi-Markov and Markov models. For a Markov model the sojourn time distributions of states must be exponentials, however, it can be any distribution for a semi-Markov model. Thus, to convert a semi-Markov to a Markov process, one must change the non-exponential distributions to exponentials. In this section, we show how to convert a state with non-exponential distribution to a number of states in which each state is exponential.

In Section 2.4 we fitted the state holding times of our resource-usage/error/recovery process to the phase-type exponentials. The phase-type exponential function $f(t)$ can be expressed as

$$f(t) = \sum_{i=1}^{n} a_i g_i(t) ,$$

where $a_i > 0$, $\sum_{i=1}^{n} a_i = 1$, and n is the number of phases. For each phase i, the $g_i(t)$ can be a simple exponential, a multi-stage hyperexponential, or a multi-stage hypoexponential. The definitions of these three types of functions are listed below.

Exponential : EXP (λ)

$$\mathrm{EXP}(\lambda) = \lambda e^{-\lambda t}$$

Hyperexponential : Hyper ($\lambda_1, \lambda_2, \ldots, \lambda_r$)

$$\mathrm{Hyper}(\lambda_1, \lambda_2, \ldots, \lambda_r) = \sum_{i=1}^{r} a_i \mathrm{EXP}(\lambda_i) = \sum_{i=1}^{r} a_i \lambda_i e^{-\lambda_i t} .$$

where $a_i > 0$, $a_i \geq 0$, and $\sum_{i=1}^{r} a_i = 1$.

Hypoexponential : Hypo $(\lambda_1,\lambda_2,....,\lambda_r)$

$$\text{Hypo}(\lambda_1,\lambda_2,....,\lambda_r) = \sum_{i=1}^{r} a_i \text{EXP}(\lambda_i) = \sum_{i=1}^{r} a_i \lambda_i e^{-\lambda_i t},$$

where $\lambda_i > 0$, $\lambda_i \neq \lambda_j$ if $i \neq j$, and $a_i = \prod_{\substack{j=1 \\ j \neq i}}^{r} \frac{\lambda_j}{\lambda_j - \lambda_i}$.

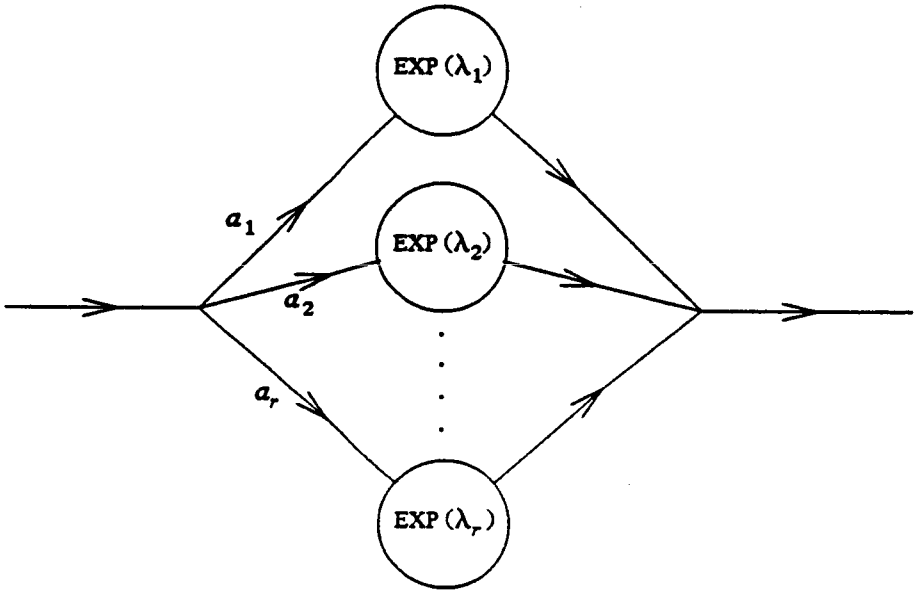
By using the method of stages [26], a hyperexponential distribution can be modeled as a set of parallel exponential stages and a hypoexponential distribution as a set of series exponential stages. Figure 5.1(a) and 5.1(b) show the conversions of these two types. In Figure 5.1(a) we note that each a_i of hyperexponential function is converted to the probability to the associated state having density $\text{EXP}(\lambda_i)$, however this is not the case for the hypoexponential. From Figure 5.1(b), we know that the Markov version of the hypoexponential is just a series connection of states in which each state has an simple exponential density function and the probability from one state to another is one. As an example we know in section 2.3 that the holding time density from state W_8 to error state DASD is fitted by

$$f(t) = 0.235\ \text{EXP}(0.004) + 0.765\ \text{Hypo}(0.00093, 0.006595),$$

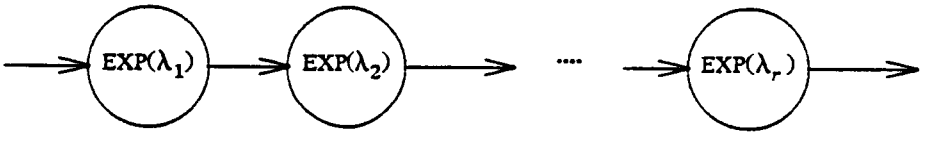
which is a combination of hyper and hypo exponentials. The Markov conversion of the state with Equation 5.2.1 holding time density is shown in Figure 5.2. Note that the state W_8 in Figure 5.2(a) is modeled as a three state Markov process. This is shown in the dotted area of Figure 5.2(b).

5.3. Performability Analysis

After converting a semi-Markov process to a Markov process, analysis can be carried out on the resulting Markov reward model of the measured system using SHARPE (the

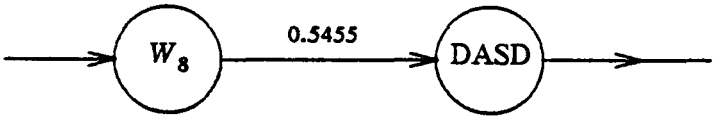


(a). Hyperexponential distribution as a set of parallel exponential stages

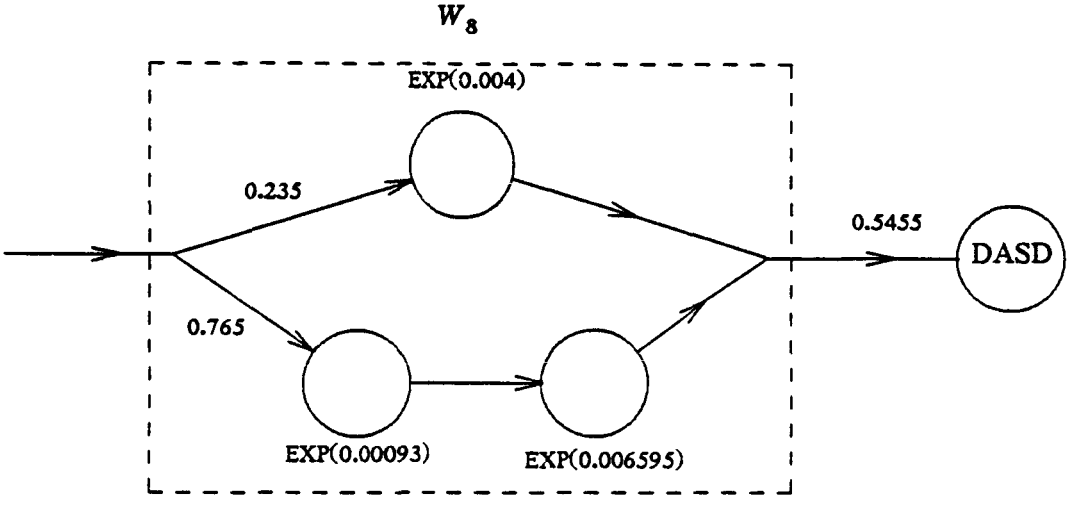


(b). Hypoexponential distribution as a set of series exponential stages

Figure 5.1. The conversion of non-exponential to a set of exponentials



(a). Semi-Markov



(b). Markov

Figure 5.2. The Markov conversion of State W_8

Symbolic Hierarchical Automated Reliability and Performance Evaluator) [29]. SHARPE is a modeling tool developed at Duke University. It provides several model types ranging from reliability block diagrams to complex semi Markov models, and allows the user to construct and analyze performance, reliability and availability models. However, this tool can only be used to analyze a model with size less then 200 states, thus we assume our

resource-usage/error/recovery model is a independent semi-Markov process. The conversion of the waiting times of states are shown in Appendix C.

In order to study the impact of different types of errors, the irreducible semi-Markov process is converted to one with absorbing states in the following manner:

a) with OFFL as the absorbing state (OFFL),

b) with MULT and OFFL as the absorbing states (MULT),

c) with SWE, MULT and OFFL as the absorbing states (SWE),

d) with DASD, MULT and OFFL as the absorbing states (DASD), and

e) with DASD, SWE, MULT and OFFL as the absorbing states (ALL).

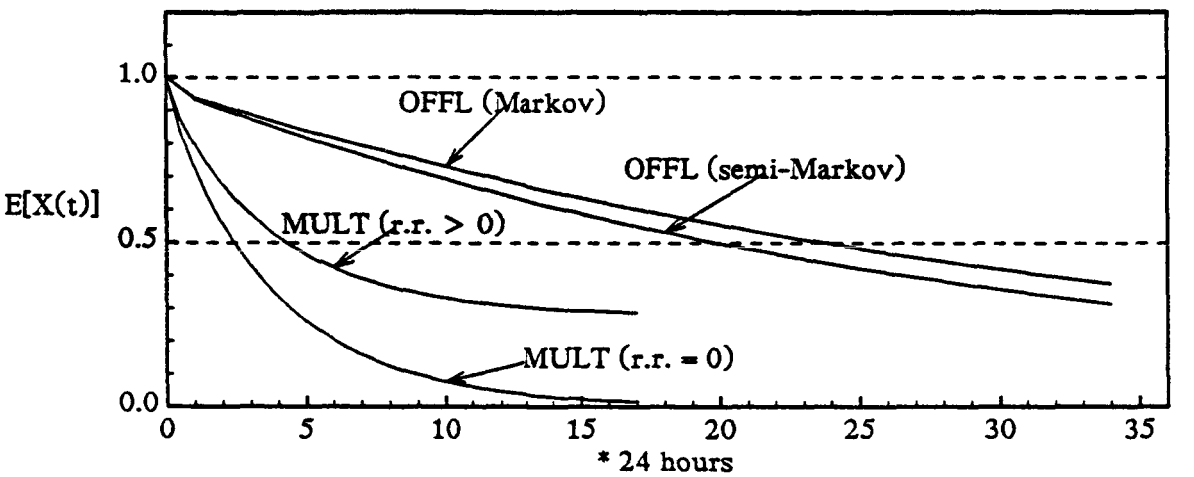
In case (a) we assess system performability in which all but off-line failures are not recovered from. This actually provides us with the result of the system reliability. In case (b) we discontinue recovering from multiple errors. Here, we expect to measure the impact on the reward to a multiple error. Since multiple errors happen much more frequently than OFFL and the sojourn time is much longer comparing with others, we expect to measure the impact of SWE and DASD on the reward to a MULT error. Thus, in case (c) we not only stop recovering from multiple and off-line failures but we also stop the recovery from software errors. In case (d) we recover from SWE errors but stop recovery from DASD errors. Finally, in case (e) we do not recover from any errors besides CHAN.

We compare these scenarios first using the expected instantaneous reward rate $E[X(t)]$ which is defined by Equation 5.1.3, then using the time-averaged expected accumulated reward $\frac{E[Y(t)]}{t}$. In all but case (a) and (e) we consider two variations: when a state such as DASD (MULT or SWE) is made absorbing, we can either let the reward rate in such a state be non-zero or we can set its reward rate to zero. The impact of the non-zero assign-

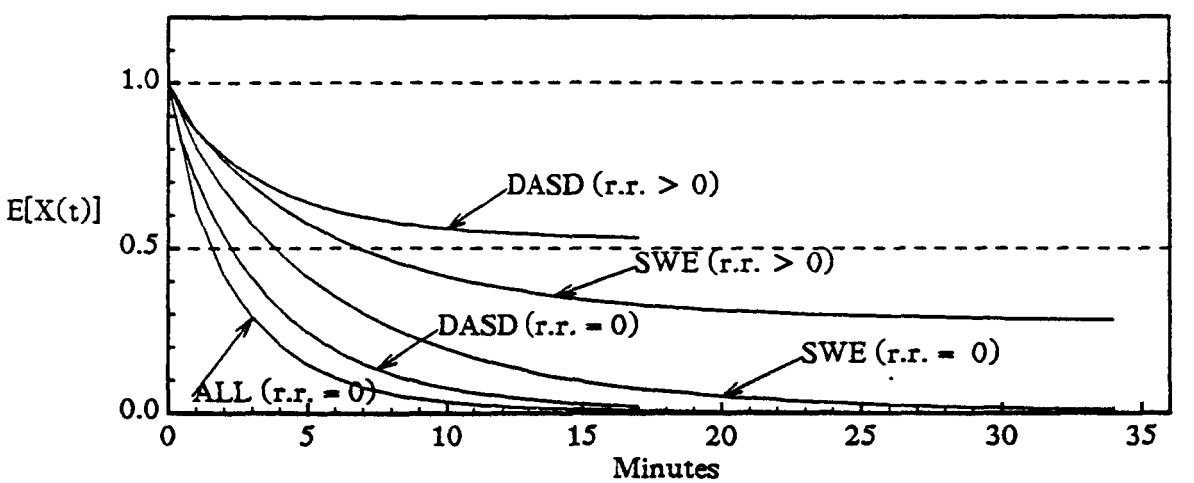
ment is that upon reaching the absorbing state, the system continues to operate in a degraded mode. In the latter case, i.e., zero reward assignment, we conservatively assume that the system stops functioning when it reaches the absorbing state(s).

In Figure 5.3(a), we plot E[X(t)] for cases (a) and (b). In the case (b) we use two different assumptions for the reward rate for the MULT state, rr_{MULT}=0.27777 and rr_{MULT}=0. We also plot E[X(t)] for case (a) with the assumption that all states have exponentially distributed holding times. We note that such a Markovian assumption leads to an overestimation of the system's capability to perform useful work, and the degree of overestimation increases as the system operating time increases. We also note that not recovering from multiple errors considerably degrades the system's performability. Moreover, changing from non-zero to zero reward rate further reduces the system's effectiveness drastically.

In Figure 5.3(b), we plot the $\frac{E[X(t)]}{t}$ for cases c, d and e. In each case, except case e, we also have two versions with reward rates for absorbing states being non-zero and zero, respectively. Note that not recovering from SWE errors degrades system effectiveness considerably compared with the effect of not recovering from DASD errors, provided we assume that absorbing states continues to provide service in a degraded mode. On the other hand, if we assume that absorbing states are system failure states, i.e., zero reward rates for absorbing states, then not recovering from DASD failures is more severe than not recovering from SWE failures. This behavior is explained by the fact that the reward rate in the DASD state is about twice that in the SWE state (0.5708 versus 0.2736 in Table 5.1). Figures 5.4(a) and 5.4(b) are the counterparts of Figures 5.3(a) and 5.3(b) where the measure plotted is $\frac{E[Y(t)]}{t}$ rather than E[X(t)]. The trends are similar.



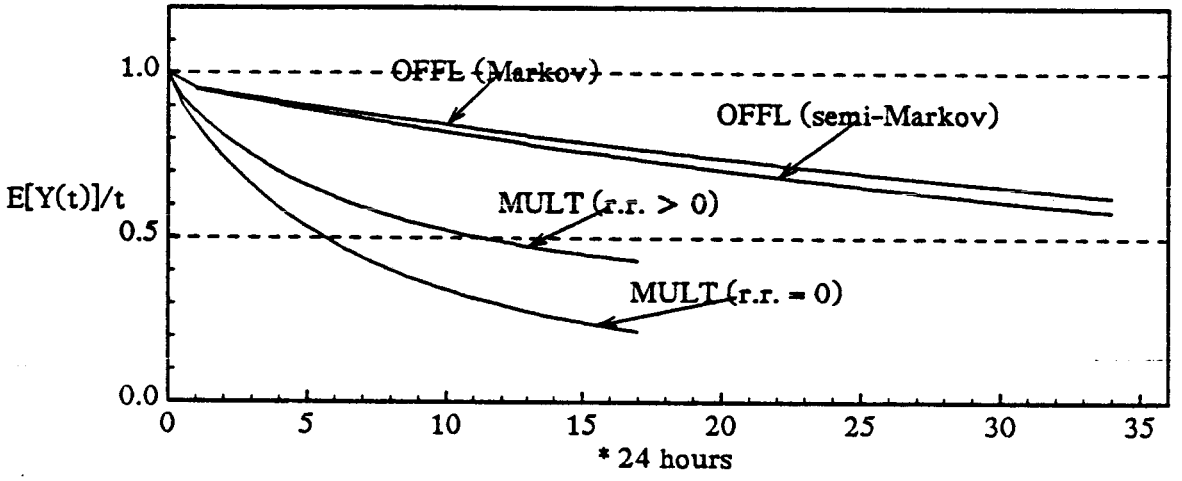
(a)



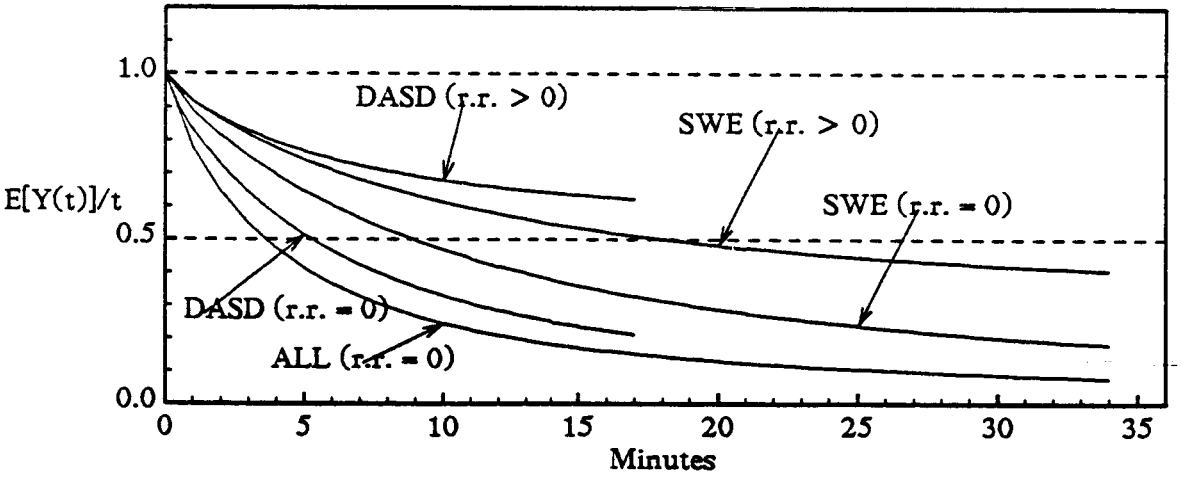
(b)

(r.r. : reward rate)

Figure 5.3. The expected reward rate, E[X(t)]



(a)



(b)

(r.r. : reward rate)

Figure 5.4. The time-averaged accumulated reward, $E[Y(t)]/t$

Finally, in Figure 5.5, we show the distribution function of $Y(\infty)$, the accumulated reward until system failure, for two cases: Markov versus semi-Markov. Both assume that the OFFL state is the only absorbing state. Once again we note that the Markovian assumption implies an overestimation of the system's performability.

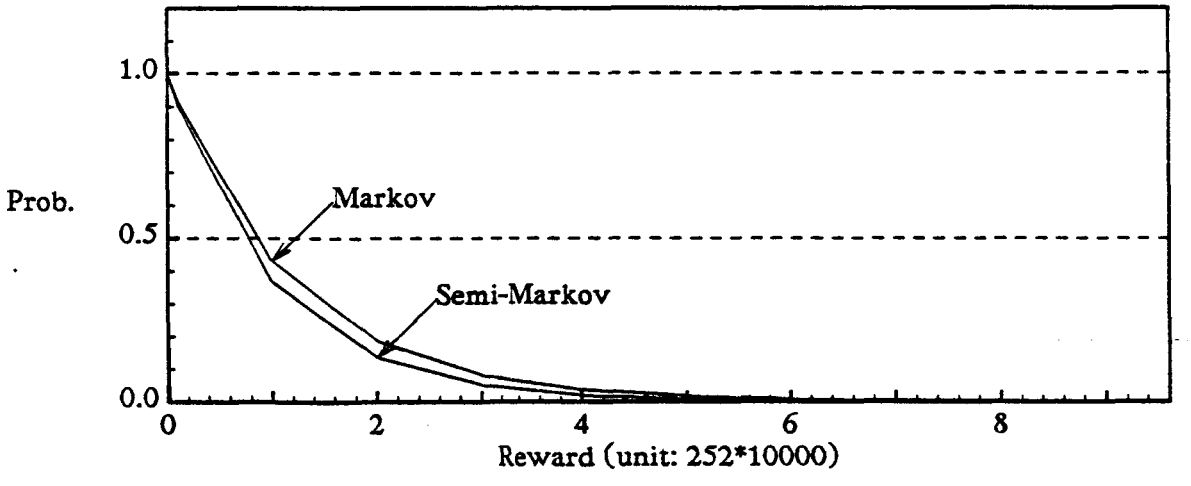


Figure 5.5. Distribution of accumulated reward until system failure

CHAPTER 6

SUMMARY AND CONCLUSIONS

6.1. Summary of Results

This thesis has developed a methodology to construct a resource-usage/reliability/performability model for a complex system based on real data. The model obtained is capable of reflecting both the normal and error behavior of the system. Both hardware and software reliability and their interactions are modeled. The effect of recovery through the built-in recovery mechanisms is also considered. By modeling the recovery process we are able to evaluate the severity of errors in general and the cost of specific error type in particular. Low-level error and resource-usage data to develop the model was collected on an IBM 3081 machine running the MVS operating system. The results of this research suggest that other production systems should be similarly analyzed so that a body of realistic data on computer error (including failure) and recovery models is available.

Chapter 2 described the development of the model, using the low level data on resource usage and errors. A statistical clustering method (k-means clustering) was employed to characterize the resource usages into a few workload clusters. A two-level error data reduction (error coalescing and grouping) scheme was used to identify individual error incidents. Results showed that about 17% of errors are multiple errors (believed to be multiple manifestations of the same problem). The state-transition diagram for a multiple

error was obtained to study the interaction between system components (hardware and software). For example, it was seen that software and disk errors were strongly correlated.

From the measurement data it was seen that the holding times in key operational and error states were not simple exponentials. A semi-Markov process was used to model the system behavior. This (semi-Markov) assumption was also validated by comparing the state occupancy probabilities predicted by the model with the actual state occupancy probabilities estimated from observed data. The results show that the proposed model provided a fairly accurate prediction of the real behavior.

The analysis of model behavior was performed in Chapter 3. The analysis showed that on-line recovery is highly effective and provides the system with the ability to tolerate many faults and recover almost instantaneously. An analysis to extract the effect of the workload on the error probability showed that not only does a higher workload result in a higher error probability (for similar holding time), but the error probability also increases with increased holding time in a particular workload state. In other words, the error probability appears to be a function of the absolute amount of resource consumed, be it through increased workload and/or increased holding times. An explanation for this "wear out" phenomenon is not clear since a large majority of the collected errors are transient, but it certainly calls into question the validity of the frequently used constant error probability assumption used in reliability modeling.

The significance of the use of a semi-Markov model, as opposed to the simple Markov model, to describe the overall resource-usage/error/recovery process was also investigated. The results showed that a simple Markov model frequently overestimates the unconditional transition probabilities and underestimates the variance of the first passage times to the error states. The overestimation can lead to an unduly conservative reliability predic-

tion and the underestimation may lead to unduly optimistic reliability prediction. Both over- and under- estimations are not desirable.

In Chapter 4, the software error data was used to build a software reliability model to describe the error and recovery processes in the MVS operating system. The semi-Markov model developed provided a quantification of the operating system error characteristics and also the interaction between different types of OS errors. We estimate that in only 0.5% of the cases the measured software system is unable to recover. A detailed model and analysis of multiple software errors, (which constitute approximately 17% of all software errors) was provided, showing how a single software problem can have multiple manifestations. To investigate the validity of this model, the duration of a multiple error predicted from the model was compared with the value estimated from the observed data. The agreement between two results was found to be within 1%.

A measurement-based performability model was discussed in Chapter 5. A reward function, based on the service rate and the error rate in each state, was proposed. In order to investigate the impact due to different errors, the expected reward rate, as well as the cumulative reward, at time t were estimated. The results show that the software error (SWE) degrades the system performance more severely than the disk error (DASD) although the error probability of DASD errors is about twice as much as that of SWE errors (0.169 versus 0.085). This may be due to the cost for DASD errors, which is less than that for SWE errors, i.e., the reward rate in DASD state is higher than that in SWE state. If, however, both error types result in system failure then, as expected, the DASD error degrades the system performance more severely than the SWE error.

The system performability under a Markov assumption is also estimated and compared with that estimated from the more realistic semi-Markov model. It was found that

the Markov assumption overestimates the system performability and that the degree of overestimation increased with increased system operation time. Once again, this indicates that the traditional Markov process is not good enough to model a computer system and to provide accurate predictions.

6.2. Suggestions for Future Research

The results of this study suggest that other systems be similarly studied so that a wide body of realistic results on computer system hardware and software performability are available. This is useful both, from the point of view of validating existing analytical models and from the point of view of generating realistic models of system behavior.

A possible extension is the area of adaptive model construction. The workload and error clustering methods employed here have potential for use in an adaptive algorithm which is capable of real-time model construction. The use of such models for adaptive tuning for optimum performability under various conditions needs to be investigated. To be successful such a system would require learning capabilities so as to use valid past information together with some knowledge of the environment for both reconfiguration under failure and for system tuning.

In this thesis we have used past data on errors and workload for model construction. It would be interesting to investigate the possibility of doing the same on the basis of data generated from error/failure injection on a prototype or into a simulation model of a system. Such a procedure has the potential of providing realistic feedback to system designers early in the development stage. A comparison of the results from such a model with those obtained through analytical models would be instructive as well.

REFERENCES

[1] Castillo, X. and Siewiorek, D.P., "A Performance-Reliability Model for Computing Systems," *IEEE*, 1980.

[2] Castillo, X. and Siewiorek, D.P., "A Workload Dependent Software Reliability Prediction Model," in *Proceedings of the 12th International Symposium on Fault-Tolerant Computing*, Santa Monica, California, pp. 279-285, June 22-24, 1982.

[3] Kulkarni, V.G., Nicola, V.F., Smith, R.M., and Trivedi, K.S., "Numerical Evaluation of Performability Measures and Job Completion Time in Repairable Fault-Tolerant Systems," in *Proceedings of the 16th International Symposium on Fault-Tolerant Computing*, Vienna, Austria, pp. 252-257, July 1-4, 1986.

[4] Meyer, J.F., "Closed-Form Solutions of Performability," *IEEE Transactions on Computers*, pp. 648-657, July 1982.

[5] Geist, R.M. and Trivedi, K., "Ultrahigh Reliability Prediction for Fault-Tolerant Computer Systems," *IEEE Transactions on Computers*, pp. 1118-1127, December 1983.

[6] Trivedi, K., Dugan, J.B., Geist, R., and Smotherman, M., "Modeling Imperfect Coverage in Fault-Tolerant Systems," in *Proceedings of the 14th International Symposium on Fault-Tolerant Computing*, Kissimme, Florida, pp. 77-82, June 20-22, 1984.

[7] Ng, Y.W. and Avizienis, A.A., "A Unified Reliability Model for Fault-Tolerant Computers," *IEEE Transactions on Computers*, pp. 1002-1011, November 1980.

[8] Goyal, A. and Tantawi, A.N., "Numerical Evaluation of Guaranteed Availability," in *Proceedings of the 15th International Symposium on Fault-Tolerant Computing*, Ann Arbor, Michigan, pp. 324-329, June 19-21, 1985.

[9] Schoen, O., "On a Class of Integrated Performance/Reliability Models based on Queuing Networks," in *Proceedings of the 16th International Symposium on Fault-Tolerant Computing*, Vienna, Austria, pp. 90-95, July 1-4, 1986.

[10] Iyer, R.K., Rossetti, D.J., and Hsueh, M.C., "Measurement and Modeling of Computer Reliability as Affected by System Activity," *ACM Transactions on Computer Systems*, vol. 4, no. 3, pp. 214-237, August, 1986.

[11] Goel, A.L., "Software Reliability Models: Assumptions and Applicability," *IEEE Transactions on Software Engineering*, vol. SE-11, pp. 1411-1423, December 1985.

[12] Yamada, S. and Osaki, S., "Software Reliability Growth Modeling," *IEEE Transactions on Software Engineering*, vol. SE-11, pp. 1431-1437, December 1985.

[13] Musa, J., "The Measurement and Management of Sofware Reliability," *IEEE Proceedings*, vol. 68, pp. 1131-1143, September 1980.

[14] Littlewood, B, "Theories of Software Reliability: How good are they and how can they be improved?," *IEEE Transactions on Software Engineering*, vol. SE-6, pp. 489-500, September 1980.

[15] Iyer, R. K. and Velardi, P., "Hardware-Related Software Errors: Measurement and Analysis," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 2, pp. 223-231, February 1985.

[16] Kelly, J. and Avizienis, A.A., "A Specification-Oriented Multi-Version Software Experiment," in *Proceedings of the 13th International Symposium on Fault-Tolerant Computing*, Milano, Italy, pp. 120-126, June 28-30, 1983.

[17] Beounes, C and Laprie, J.C., "Dependability Evaluation of Complex Computer Systems: Stochastic Petri Net Modeling," in *Proceedings of the 15th International Symposium on Fault-Tolerant Computing*, Ann Arbor, Michigan, pp. 364-369, June 19-21, 1985.

[18] Castillo, X., *A Compatible Hardware/Software Reliability Prediction Model*. PhD Thesis, Carnegie-Mellon University, July, 1981.

[19] IBM Corperation, "MVS Architecture Resource Measurement Facility (RMF) Reference and User's Guide," *IBM publications*, vol. LC28-1138-1.

[20] Ferrari, D., Serazzi, G., and Zeigner, A., *Measurement and Tuning of Computer Systems*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.

[21] MacQueen, J., "Some Methods for Classification and Analysis of Multivariate Observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, Berkeley, California, pp. 281-297, June 21-July 18, 1965.

[22] Spath, H, *Cluster Analysis Algorithms*. West Sussex, England: Ellis Horwood Ltd., 1980.

[23] IBM Corperation, *Environmental Record Editing & Printing Program*. International Business Machines Corporation, 1984.

[24] Iyer, R.K., Young, L.T., and Sridhar, V., "Recognition of Error Symptoms in Large Systems," in *Proceedings of the 1986 IEEE-ACM Fall Joint Computer Conference*, Dallas, Texas, pp. 797-806, November 2-6, 1986.

[25] Howard, Ronald A., *Dynamic Probabilistic Systems*. New York: John Wiley & Sons, Inc.,, 1971.

[26] Trivedi, K.S., *Probability & Statistics with Reliability, Queuing, and ComputerScience Applications*. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1982.

[27] Hsueh, M.C., Iyer, R.K., and Trivedi, K.S., "A Measurement-Based Performability Model for a Multiprocessor System," in *Proceedings of the 2nd International Workshop on Applied Mathematics and Performance/Reliability Models of Computer/Communication Systems*, Rome, Italy, May 25-29, 1987.

[28] Smith, R.M., Trivedi, K.S., and Nicola, V.F., "The Analysis of Computer Systems Using Markov Reward Processes," 1987.

[29] Sahner, R.A. and Trivedi, K.S., "SHARPE: Symbolic Hierachical Automated Reliability and Performance Evaluator," in *User's Guide*, Durham, NC, September 1986.

APPENDIX A

(I). Error Clustering

Identical errors occurring within 5 minutes of each other were coalesced into a single event. This was done to ensure that the analysis is not biased by failure records relating to the same problem. The clustering algorithm analyzes the data and merges observations which occur in rapid succession and relate to the same problem. For each failure point, the following test was performed :

IF <error type> = <type of previous error> AND
<time away from previous error> $\leqslant$ 5 minutes
THEN
<fold error into cluster being built>
ELSE
<start a new cluster>

The result is a set of clustered errors. Associated with each cluster is information consisting of error classifications, number of points in the cluster, time of first and last errors in the cluster, and a variety of status data provided by the hardware and operating system.

(II). Error Grouping

A visual examination of the error clusters showed the existence of sets of clusters occurring within a short time interval. The close time proximity among some clusters means a substantial increase in the system error rate during that period. The high error

rate introduces the suspicion that the errors occurring during the high error rate period may be related, i.e., different errors may be due to a single cause, to multiple but related causes, or to multiple and independent causes. Therefore, the high error rate periods are formed by grouping all error clusters occurring within a small time interval of each other. This interval was chosen to be 5 minutes. The result is a set of grouped errors. The primary difference between a cluster and a group is that clusters contain only occurrences of the same error (same error type and machine state), whereas groups contain occurrence of different errors (different error type or machines state).

APPENDIX B

The Characteristics of the Resource-Usage/Error/Recovery Model

(I). Stochastic transition probability matrix.

(Due to the size of the matrix, it is broken into two parts, (a) and (b).)

	W_0	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8
W_0	0	0.351	0.055	0.108	0.135	0	0.216	0.027	0.108
W_1	0.019	0	0	0.057	0.170	0	0.113	0.132	0.019
W_2	0	0	0	1	0	0	0	0	0
W_3	0.143	0.048	0	0	0.143	0.048	0.048	0.048	0
W_4	0.023	0.045	0	0	0	0.015	0.046	0.099	0.038
W_5	0.091	0	0	0	0	0	0.091	0	0.091
W_6	0.034	0.034	0	0.013	0.054	0	0	0.087	0.067
W_7	0.040	0.011	0	0	0.022	0	0.051	0	0.069
W_8	0.093	0	0	0	0.007	0	0.015	0.063	0
CPU	0	0	0	0	0	0	0	0	0
CHAN	0	0	0	0	0	0	0	0	0
SWE	0	0	0	0	0	0	0	0	0
DASD	0	0	0	0	0	0	0	0	0
MULT	0	0	0	0	0	0	0	0	0
HWR	0.018	0.036	0	0.014	0.013	0.013	0.144	0.314	0.329
SWR	0.006	0.046	0	0.008	0.160	0.007	0.183	0.286	0.306
ALT	0	0	0	0	0	0	0	0	0
OFFL	1	0	0	0	0	0	0	0	0

(a)

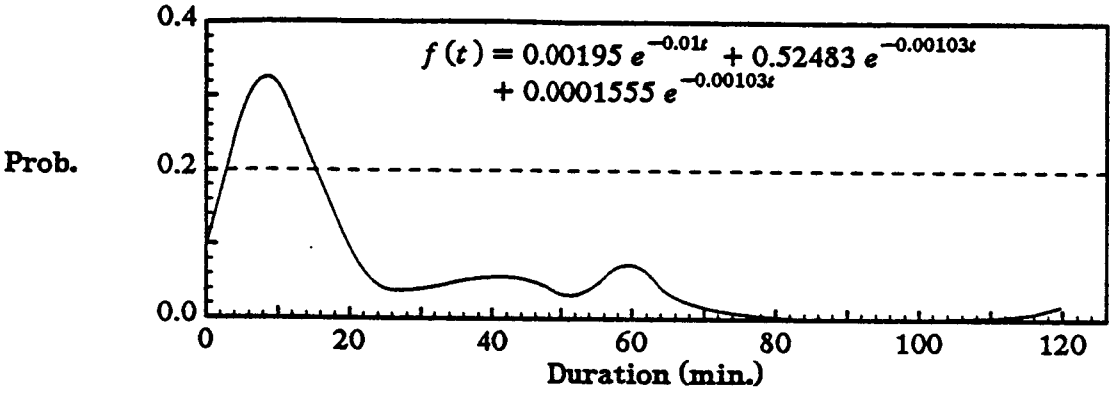
C-2

	CPU	CHAN	SWE	DASD	MULT	HWR	SWR	ALT	OFFL
W_0	0	0	0	0	0	0	0	0	0
W_1	0	0.019	0.151	0.226	0.094	0	0	0	0
W_2	0	0	0	0	0	0	0	0	0
W_3	0	0	0.143	0.381	0	0	0	0	0
W_4	0	0.015	0.227	0.386	0.106	0	0	0	0
W_5	0	0	0.091	0.545	0.091	0	0	0	0
W_6	0	0.007	0.262	0.383	0.060	0	0	0	0
W_7	0	0.026	0.208	0.482	0.091	0	0	0	0
W_8	0	0.007	0.231	0.502	0.082	0	0	0	0
CPU	0	0	0	0	0	1	0	0	0
CHAN	0	0	0	0	0	1	0	0	0
SWE	0	0	0	0	0	0.5	0.5	0	0
DASD	0	0	0	0	0	1	0	0	0
MULT	0	0	0	0	0	0.643	0.355	0	0
HWR	0	0	0	0	0	0	0.017	0	0
SWR	0	0	0	0	0	0	0	0	0.003
ALT	0	0	0	0	0	0	0	0	0
OFFL	0	0	0	0	0	0	0	0	0

(b)

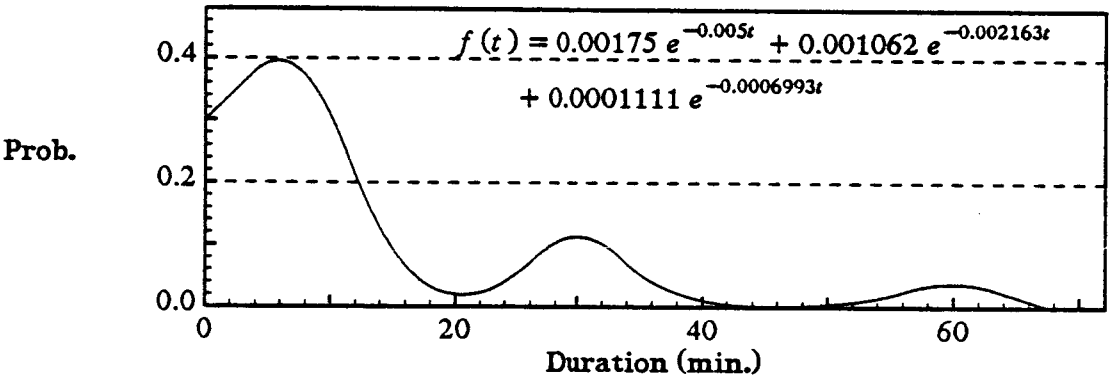
(II). Waiting and holding time densities. Constant sojourn times are assigned in W_0, W_2, CPU, CHAN and recovery states. The others are shown below.

Waiting time density of W_1



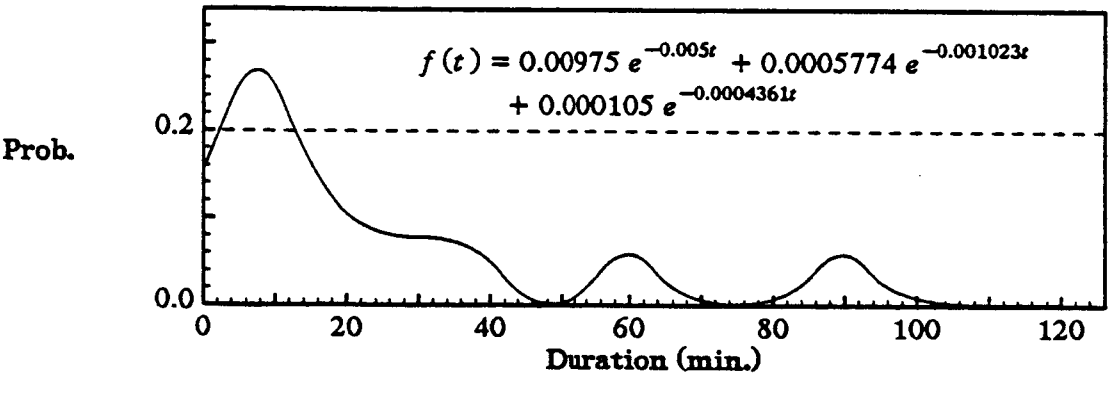
Waiting time density of W_1

Waiting time density for W_3



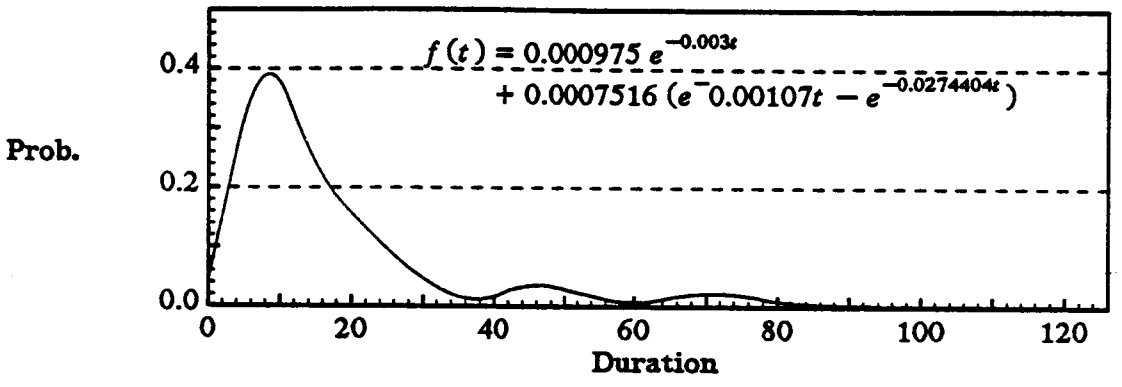
Waiting time density for W_3

Waiting/holding time densities for W_4

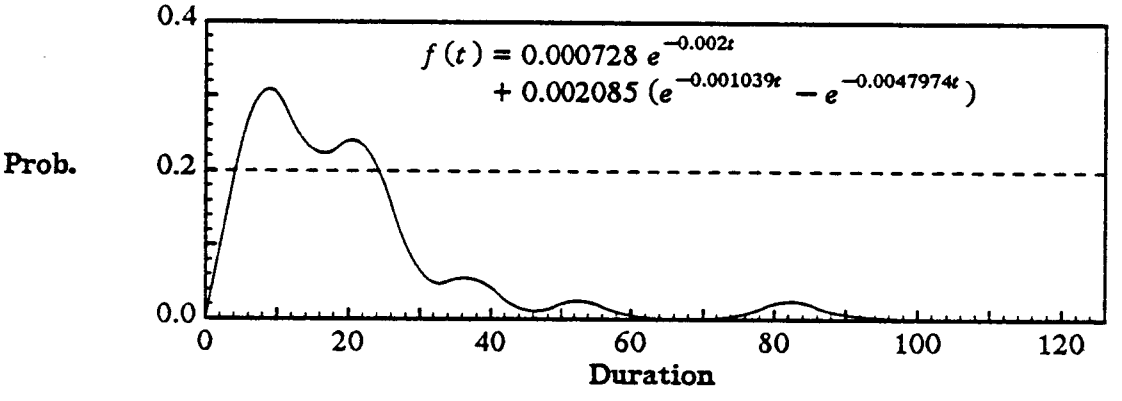


Holding time density to W_7

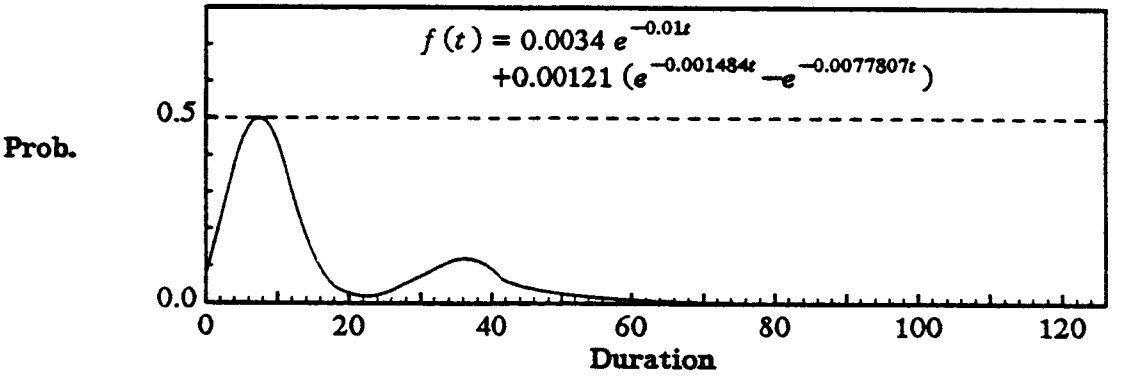
Waiting/holding time densities for W_4



Holding time density to DASD

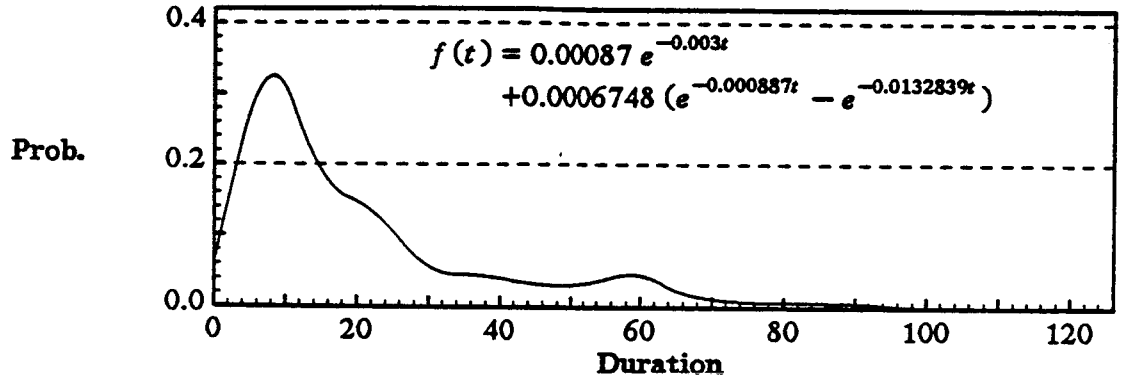


Holding time density to SWE



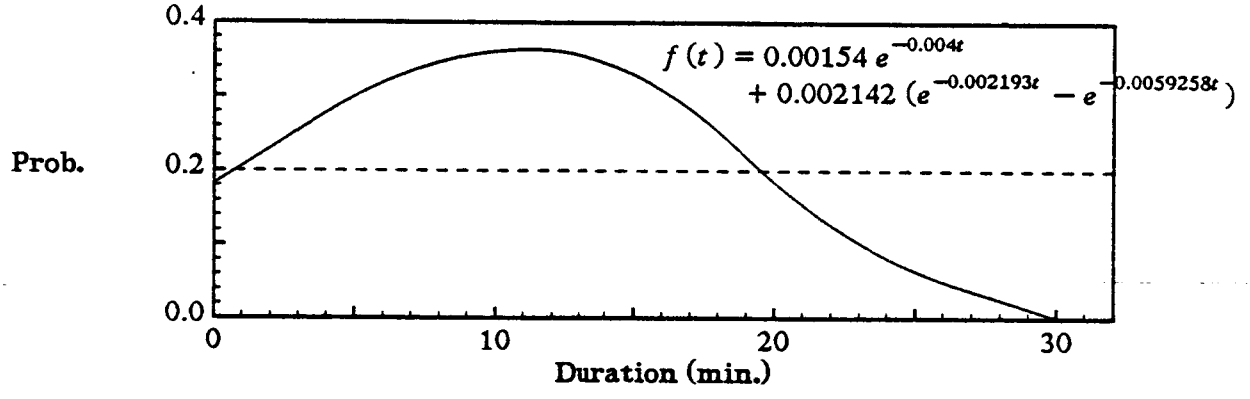
Holding time density to MULT

Waiting/holding time densities for W_4



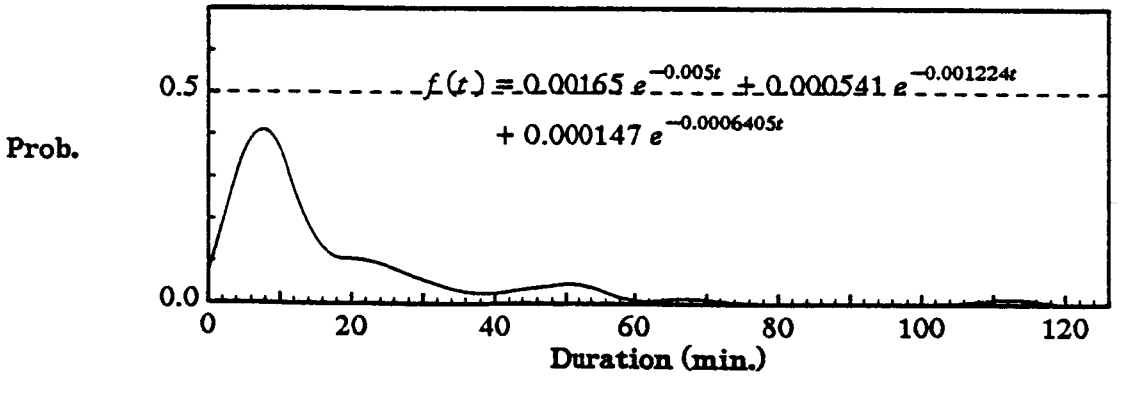
Holding time density to the others

Waiting time density of W_5



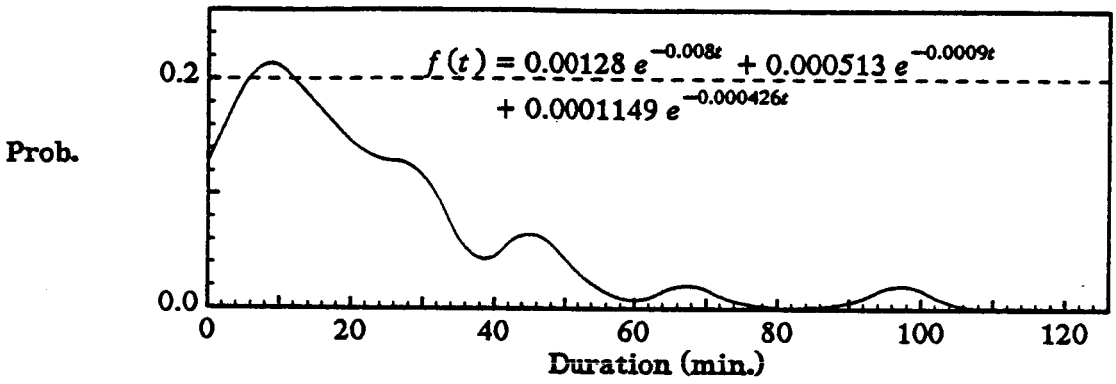
Waiting time density of W_5

Waiting/holding time densities for W_6

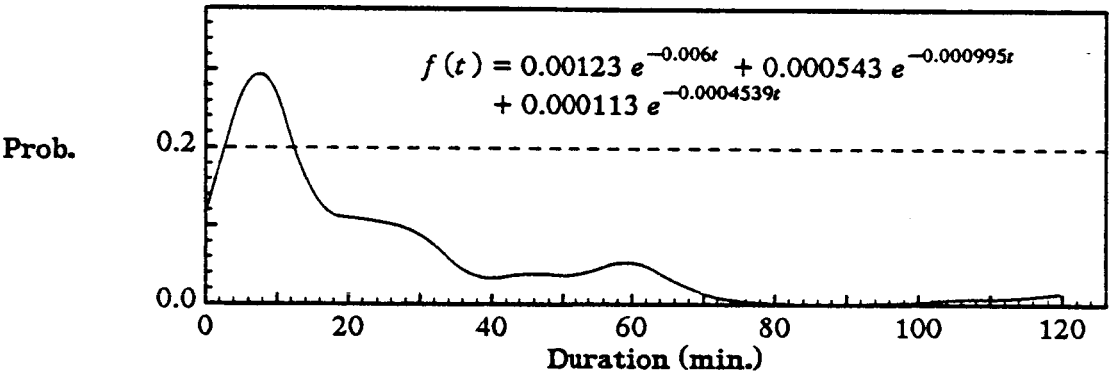


Holding time density to DASD

Waiting/holding time densities for W_6

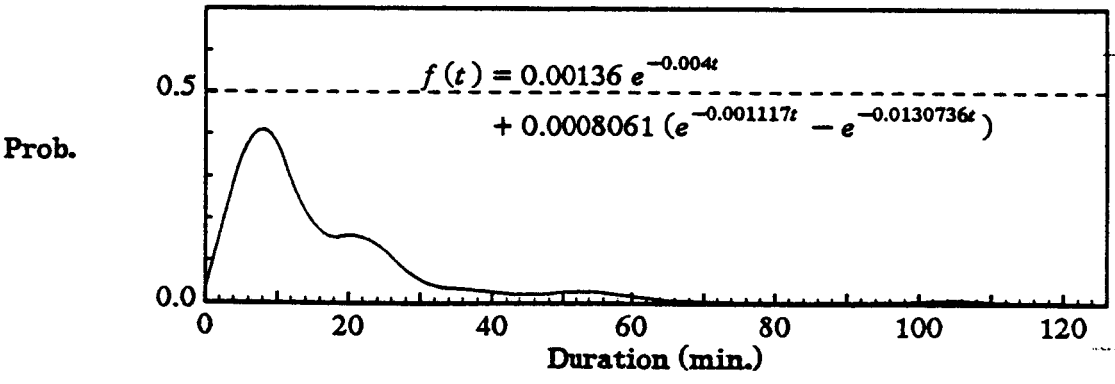


Holding time density to SWE



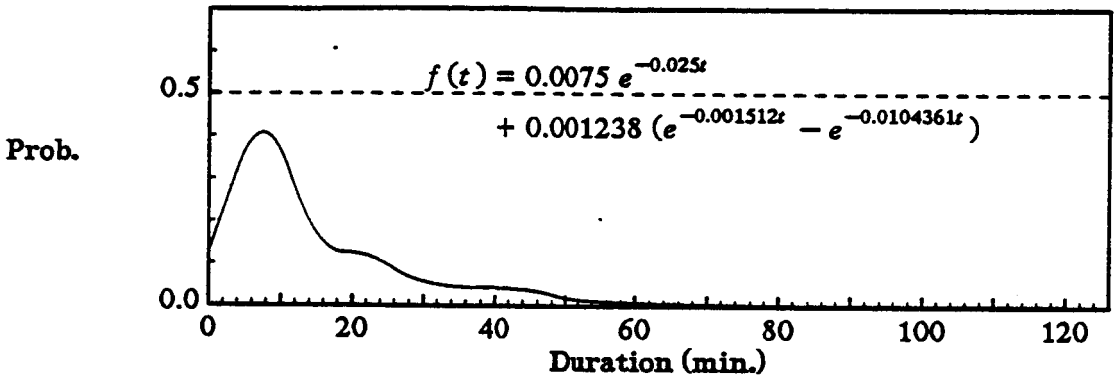
Holding time density to the others

Waiting/holding time densities for W_7

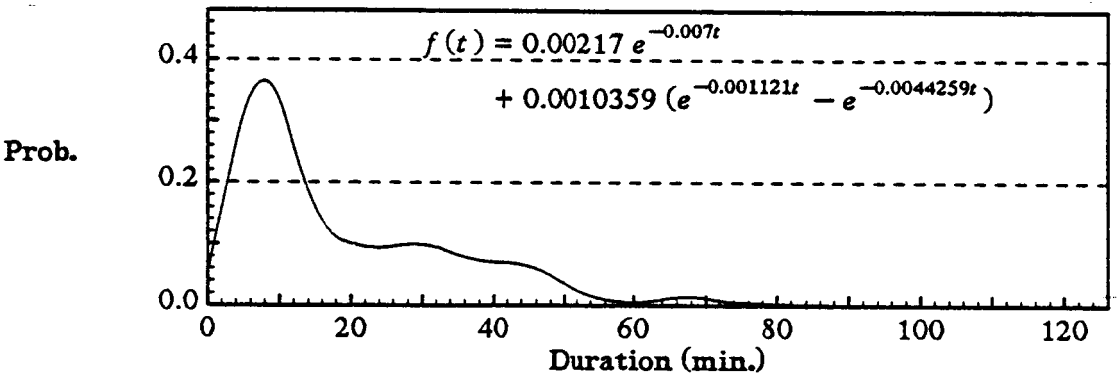


Holding time density to DASD

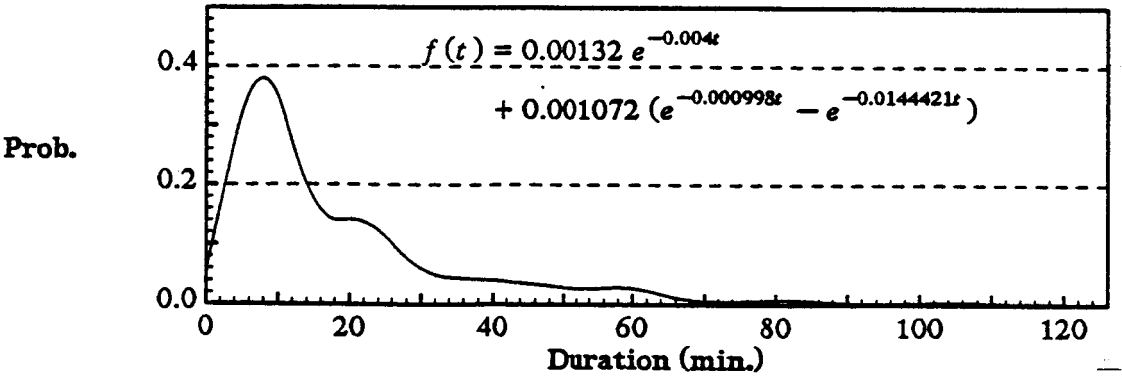
Waiting/holding time densities for W_7



Holding time density to MULT

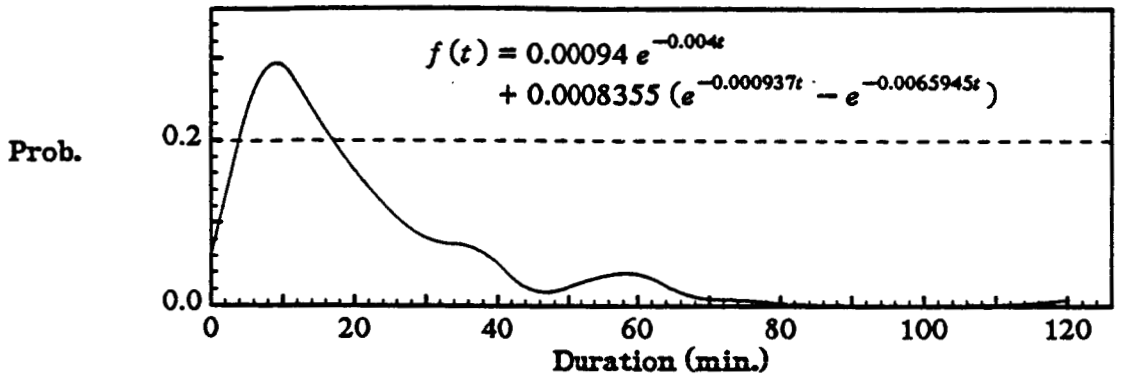


Holding time density to SWE

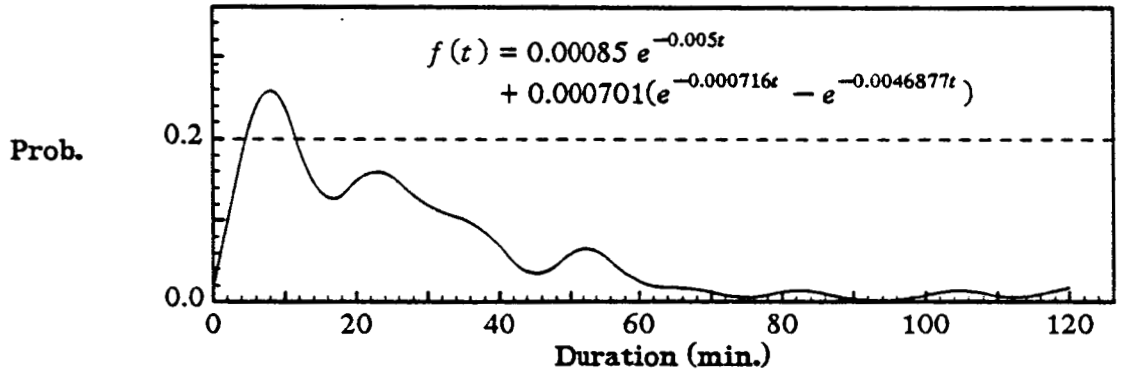


Holding time density to the others

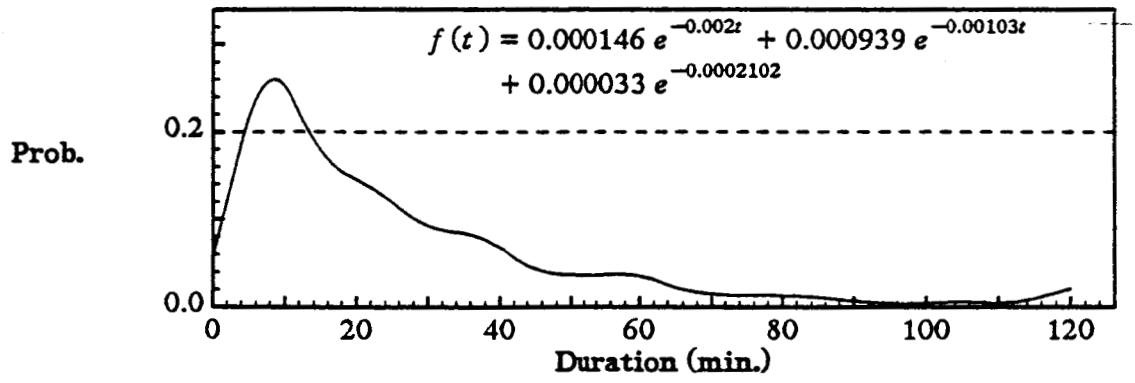
Waiting/holding time densities for W_8



Holding time density to DASD

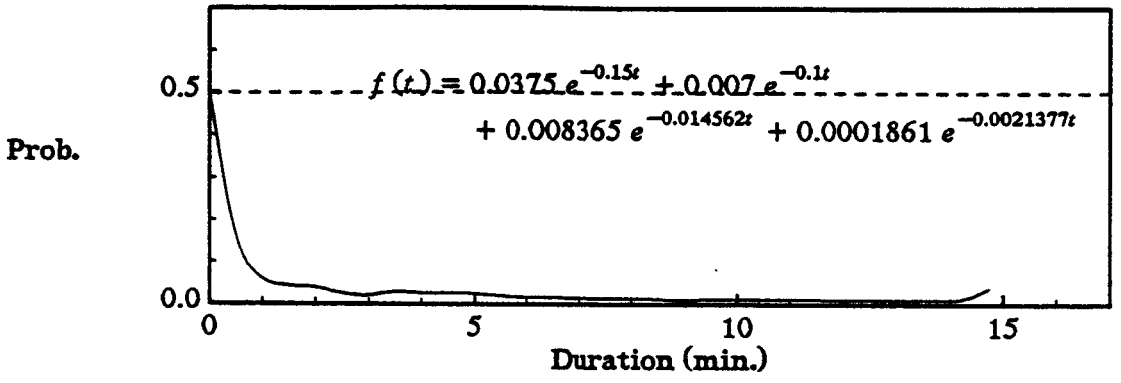


Holding time density to SWE

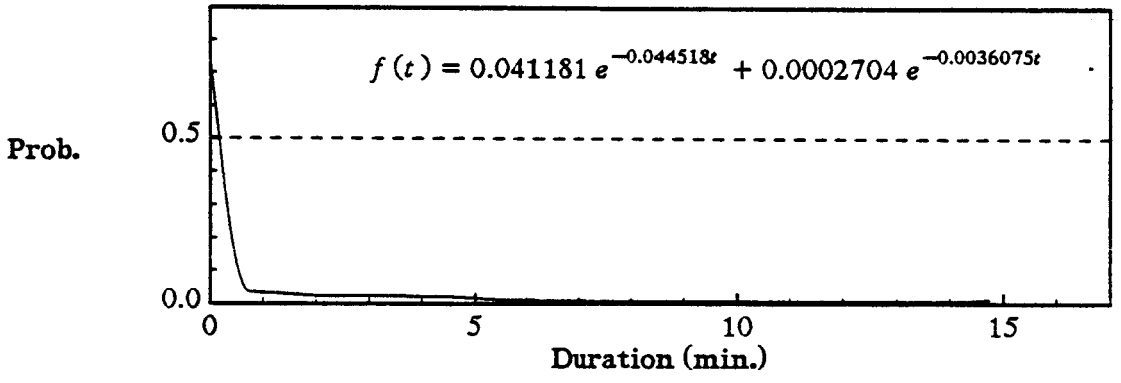


Holding time density to the others

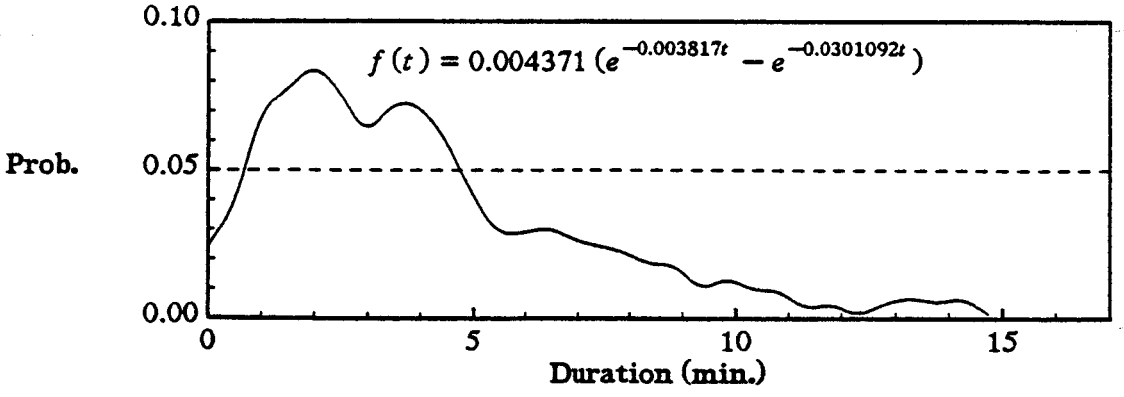
Waiting time densities for Error States



Waiting time density of DASD



Waiting time density of SWE



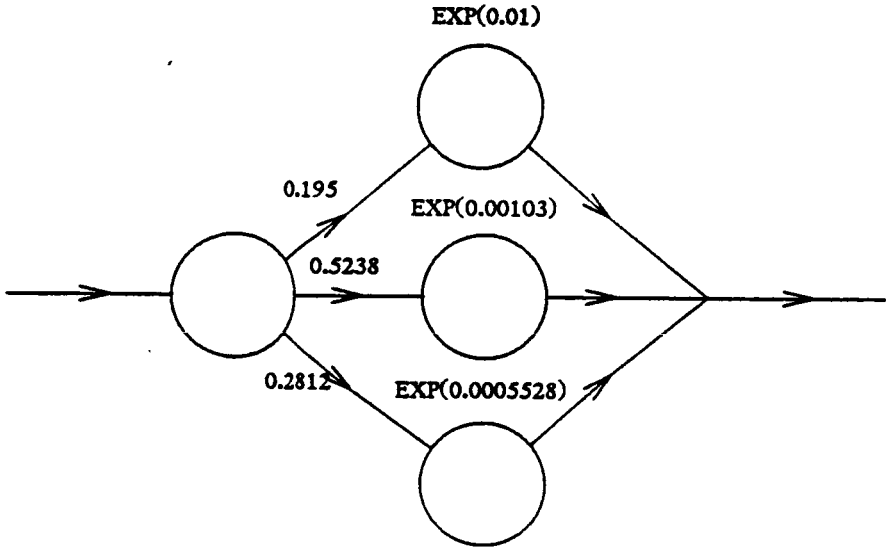
Waiting time density of MULT

APPENDIX C

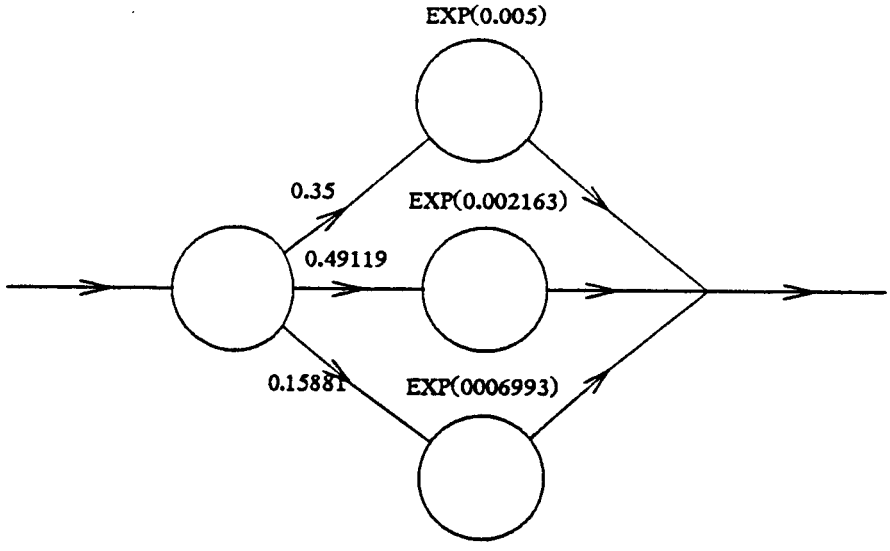
Semi-Markov to Markov Conversion

The state conversions of the resource-usage/error/recovery process from a semi-Markov model to Markov model is demonstrated in this appendix. Here, we assume that the model is an independent semi-Markov process because of the limitation of SHARPE.

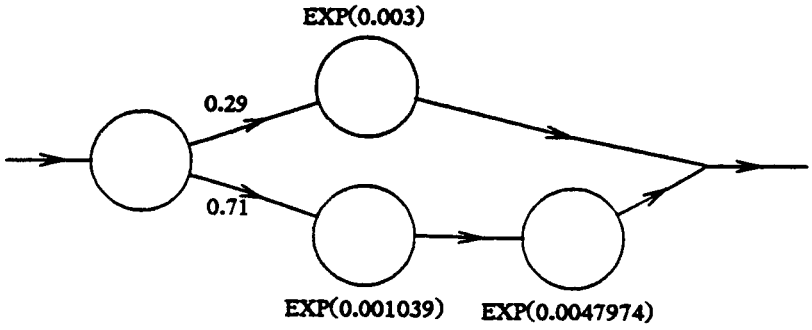
The CPU bound workload state is used to estimate the system's performability. State W_2 is combined with W_3 because W_2 has very few observations. CHAN error state is also ignored because it has very few observations. The semi-Markov to Markov conversion of the workload states are shown from Figure (a) through (g) and the conversion of three error states are shown from Figure (h) through (j). After the conversion, the overall model is expanded from 15 states to 34 states.



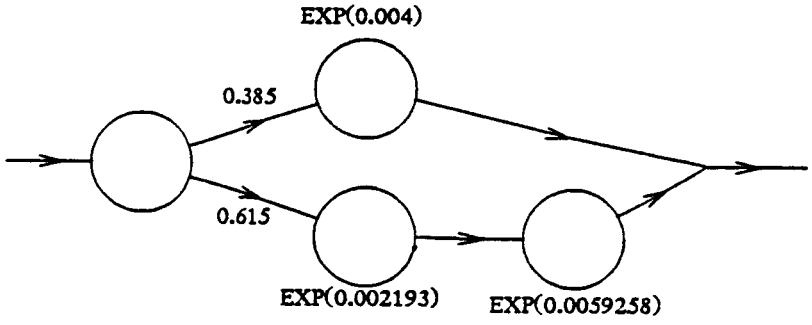
(a). W_1 state



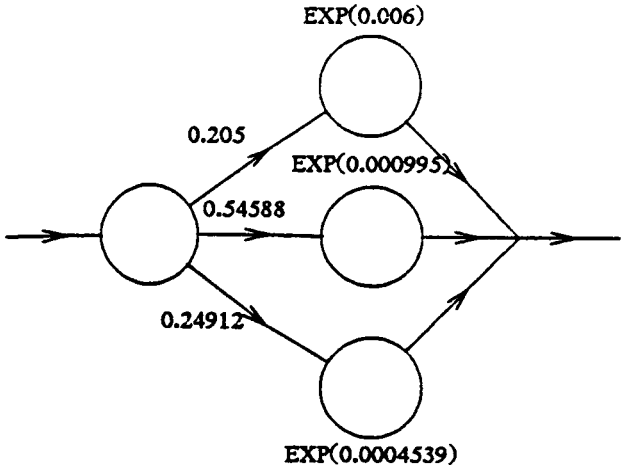
(b). W_3 state



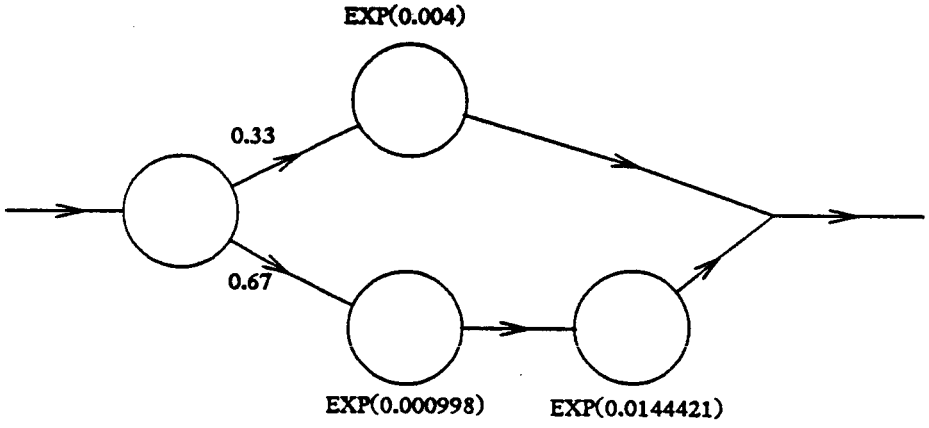
(c). W_4 state



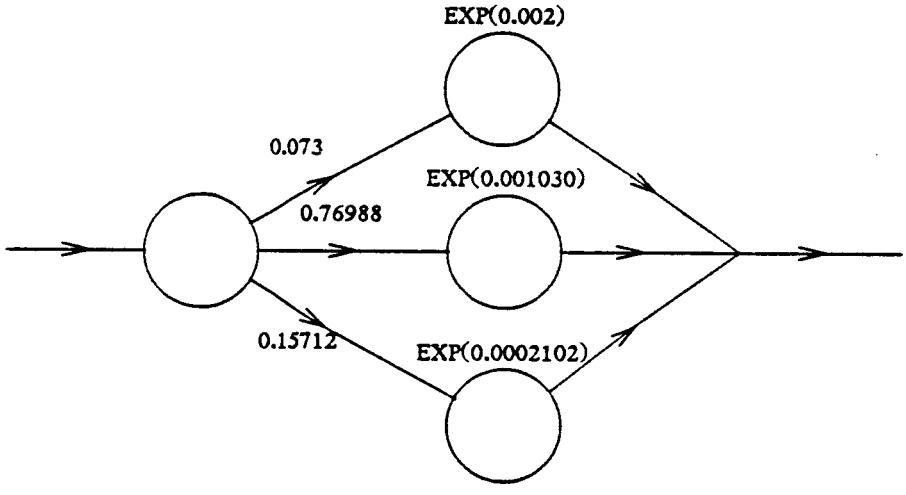
(d). W_5 state



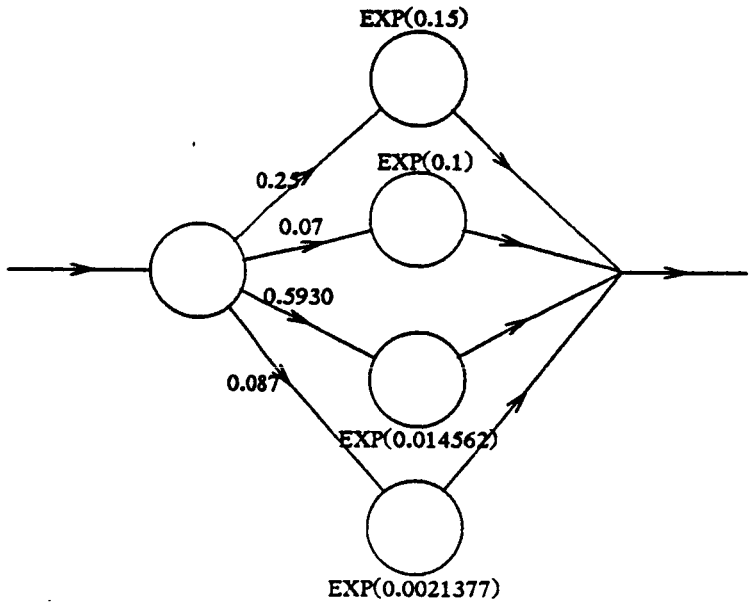
(e). W_6 state



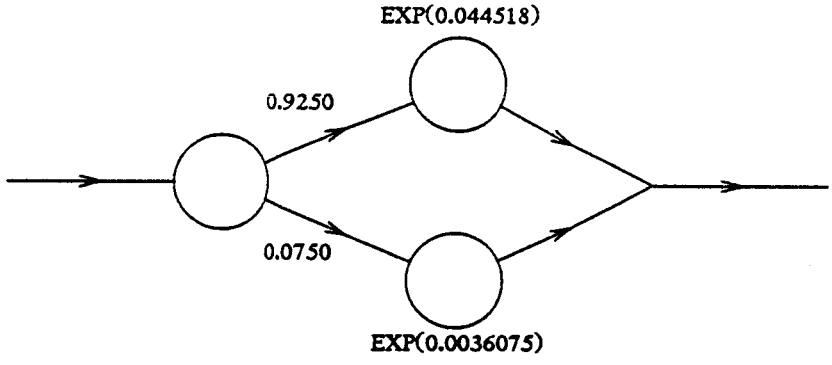
(f). W_7 state



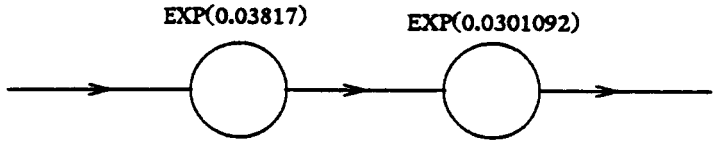
(g). W_8 state



(h). DASD state



(i). SWE state



(j). MULT state

VITA

Mei-Chen Hsueh was born on [redacted] in [redacted]. She received her B.S. degree and M.S. degree both in Mathematics from Providence College, Taiwan, 1972 and Portland State University, Portland, Oregon, 1981, respectively. She obtained her Ph.D. degree in Computer Science from the University of Illinois at Urbana-Champaign, 1987.

She joined the Department of Industrial Engineering, Fen-Chia University as a full time teaching assistant from 1972 to 1974 and worked as a quality assurance engineer for Ampex Corporation, Taiwan, from 1975 to 1979. While attending Portland State University she worked as a teaching assistant for the Department of Mathematics and as a programming consultant for the Computer Center. After graduation in 1981, she joined Floating Point Systems, Inc. as a software engineer and left as a senior software engineer in 1983. As a graduate student in the University of Illinois, she was a research assistant in both the Department of Political Science (1983-1984) and the Computer Systems Group, Coordinated Science Laboratory (1984-1987).

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-87-2258 (CSG-71)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION NASA, IBM, and ONR
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Avenue Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) NASA Langley Research Ctr. Hampton, VA 23665 IBM: T. J. Watson Research Center P.O. Box 218 Yorktown Heights, NY 10598 (over)
8a. NAME OF FUNDING / SPONSORING ORGANIZATION NASA, IBM, and JSEP 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER NASA: NAG-1-613 JSEP: N00014-84-C-0149
8c. ADDRESS (City, State, and ZIP Code) See block 7b.	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Measurement-Based Reliability/Performability Models

12. PERSONAL AUTHOR(S)
Hsueh, Mei-Chen

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	September 1987	110

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	reliability, measures, research usage, failure, performability, semi-Markov, phase-type explanation, reward

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report describes measurement-based models based on real error-data collected on a multi-processor system. Models development from the raw error-data to the estimation of cumulative reward is described.

A workload/reliability model is developed based on low-level error and resource usage data collected on an IBM 3081 system during its normal operation in order to evaluate the resource-usage/error/recovery process in a large mainframe system. Thus, both normal and erroneous behavior of the system are modeled. The results provide an understanding of the different types of errors and recovery processes. The measured data show that the holding times in key operational and error states are not simple exponentials and that a semi-Markov process is necessary to model the system behavior. A sensitivity analysis is performed to investigate the significance of using a semi-Markov process, as opposed to a Markov process, to model the measured system.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

7b. Address (continued)

800 N. Quincy St.
Arlington, VA 22217 (ONR)

19. Abstract (continued)

A software reliability model is also developed based on low-level error data from the MVS operating system running on an IBM 3081 machine to describe the software error and recovery process. The semi-Markov model developed provides a quantification of system error characteristics and the interaction between different types of errors. As an example, we provide a detailed model and analysis of multiple errors, which constitute approximately 17 percent of all software errors and result in considerable recovery overhead. In addition, a measurement-based performability model based on real error-date collected is proposed. A reward function, based on the service rate and the error rate in each state, is defined in order to estimate the performability of the system and to depict the cost of different error types and recovery procedures.